DTIC FILE COPY

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

AD-A220 357

AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1990	Final Report, 15 Sep 89 to 14 Dec 89

TITLE AND SUBTITLE
INTELLIGENT REAL-TIME PROBLEM SOLVING

5. FUNDING NUMBERS
F49620-89-C-0129
62707F 5581/A7

AUTHOR(S)
PROFESSOR Jay Lark

PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cimflex Teknowledge Corp.
1810 Embarcadero Road
P. O. Box 10119
Palo Alto, CA 94303

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR 90-0323

SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Bldg. 410
Bolling AFB, DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
f49620-89-C-0129

DTIC
ELECTE
MAR 29 1990
S D
D

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A workshop on Intelligent Real-Time Problem Solving (IRTPS) was held in Santa Cruz, California, November 6 and 7, 1989. The workshop was sponsored by AFOSR (Air Force Office of Scientific Research), RADC (AF Rome Air Development Center), and WRDC (AF Wright Research and Development Center) as part of an initiative to stimulate the development of a national basic research focus on IRTPS. This report summarizes the results of that workshop and the work leading up to it.

14. SUBJECT TERMS	15. NUMBER OF PAGES	16. PRICE CODE
	XI-1	

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTELLIGENT REAL-TIME PROBLEM SOLVING (IRTPS): Workshop Report

November 8/9, 1989
Santa Cruz, CA

This report produced for the Air Force by:

Cimflex Teknowledge Corp.
1810 Embarcadero Road
P. O. Box 10119
Palo Alto, CA 94303

Report TTR-ISE-90-101

Lee D. Erman (editor)

January, 1990

Research sponsored by the
Air Force Office of Scientific Research (AFSC),
under Contract F49620-89-C-0129 and other contracts.

Table of Contents

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

QUALITY INSPECTED 4

Preface

The state of the art in real-time AI systems lags far behind the needs of the applications requiring this technology. Current researchers view this technology as being at least one generation behind the expectations for real-time applications, the computational requirements of which are increasing in complexity faster than the speed of today's processors. Current problem-solving technology does not meet the expected stringent requirements for predictable and high quality results achieved in a timely manner in the presence of excessive demands for resources, where response time and computing resources are limited and varying, focus of attention shifts frequently, large amounts of information must be managed under severe time constraints, data and knowledge uncertainties exist, and goals conflict.

Many current and proposed programs throughout DoD and the Air Force will require Intelligent Real-Time Problem Solving (IRTPS) capabilities. As a consequence, the Air Force needs to increase the national focus on IRTPS throughout the computer science community, including those segments involved with AI approaches to problem solving, software engineering for traditional real-time systems, decision analysis, and control theory.

Scientists and engineers at Wright Research and Development Center (WRDC), at Rome Air Development Center (RADC), and at institutions supported by grants and contracts from the Air Force Office of Scientific Research (AFOSR) are seeking and developing technological advances needed to help Air Force commanders cope with the ever increasing complexities of military conflict. To support air crews, research and development efforts are underway at WRDC to automate many mission functions, such as mission planning and replanning, tactics planning, target detection and identification, terrain following and avoidance, navigation, threat avoidance, sensor fusion, and situation assessment. To support Air Force CINC's in their battle management, research and development activities are underway at RADC, including crisis management tasks across the full spectrum of command, control, communications, and intelligence functions.

In addition to these efforts, AFOSR, RADC, and WRDC have set aside sufficient resources for the following four phases of planned activity.

1. Define terms and issues.
2. Define paradigmatic IRTPS problems.
3. Generate approaches and solutions, and conduct experiments.
4. Compare solutions and approaches, and disseminate results.

Phase 1 resulted in a survey and analysis of the many manifestations of IRTPS and the circumscription of a central class of IRTPS issues.

Phase 2 is represented by this document. It contains a concise definition of the terms and issues that adequately describe IRTPS as a domain of fundamental scientific inquiry. This will serve as a focus of research for subsequent phases of this program and beyond.

Phase 3, which is to follow the dissemination of this report, will include the competitive identification of three to ten research teams, each of which will be charged with an eighteen month development of research ideas and techniques to solve IRTPS test problem(s). The result

of Phase 3 will be a final technical report documenting each team's approach to IRTPS and the progress made, and running prototype systems that embody those techniques.

Phase 4 will consist of a second scientific workshop, attended mainly by representatives of each research team from Phase 3. Results from Phase 3 will be "flown off" against each other, and individual and panel discussions will focus on IRTPS fundamental research results achieved during Phase 3. An assessment of the remaining obstacles and directions for future IRTPS research will be a major topic for consideration.

Phase 4 will be documented by a short, quickly appearing research article describing the workshop results and a larger, more significant in-depth scientific report suitable for scholarly publication that will provide critical technical assessment of and prognosis for IRTPS. It will also serve as the primary initial scientific reference for future IRTPS research activities.

Abe Waksman (AFOSR)
Bill Baker (WRDC)
Nort Fowler (RADC)

Editor's/Organizer's Acknowledgments

I would like to thank all those who contributed to the workshop, the preliminary work, and this report. Such a fine group of people made my work pleasurable.

Abe Waksman of AFOSR shepherded the entire effort; he made most of the bureaucratic details disappear and always managed to keep the scientific, technical, and community concerns paramount. Nort Fowler (RADC), and Bill Baker and Mike Wellman (WRDC) also provided much helpful guidance. The various team players responded well to insistent demands and many even kept to schedule; they *always* produced interesting scientific ideas and discussion: Bruce D'Ambrosio, Paul Cohen, Michael Fehling, Matt Ginsberg, Barbara Hayes-Roth, Eric Horvitz, Adele Howe, Stan Rosenschein, and Yoav Shoham.

All of the workshop attendees contributed with their ideas, spirit, and good will. Special thanks to the following for volunteering and following through on special and important roles: Tom Dean, Ed Durfee, Armen Gabrielian, Les Gasser, Kurt Konolige, Ted Linden, Paul Rosenbloom, Stuart Russell, and N. S. Sridharan.

At Cimflex Teknowledge, Rick Hayes-Roth and Jay Lark contributed to the technical directions of the workshop planning. JoAnn Green helped with much of the preparation and subsequent work. Charisse Arenas handled the workshop logistics and kept things running smoothly at the workshop itself so that I was able to participate fully in the substance of the two days.

Sections 2 through 5 of this report were each written by the respective working-group leaders/editors, as noted at the start of each section. They attempted to capture the sense of the group, but do not mean to imply that each member of the group (or even all of the editors for the group) subscribed to all points indicated. Or, as Bruce D'Ambrosio put it, "Anyone who thinks that all working group members agree with everything in this report has never been a member of a working group."

--- L.D.E.

1. Introduction

A workshop on Intelligent Real-Time Problem Solving (IRTPS) was held in Santa Cruz, California, November 6 and 7, 1989. The workshop was sponsored by AFOSR (Air Force Office of Scientific Research), RADC (AF Rome Air Development center), and WRDC (AF Wright Research and Development Center) as part of an initiative to stimulate the development of a national basic research focus on IRTPS. This report summarizes the results of that workshop and the work leading up to it.

1.1 Scientific and Technical Context

Despite continuing progress in research and applications of knowledge systems, current technology does not meet the stringent requirements for predictable, high-quality results achieved in a timely manner in the presence of excessive demands for problem-solving resources. Recently, research has begun to extend knowledge systems technology to include real-time interactions with ongoing processes. This requires capabilities in several dimensions not addressed by current knowledge systems technology. These include

- ensuring that the application will produce relevant output in a time appropriate for the environment,
- executing the functionalities continuously,
- adapting the reasoning process and its guiding strategies to changes in the environment that invalidate previous input or modify the available problem-solving resources,
- interacting asynchronously, and gracefully, with the environment,
- interrupting ongoing reasoning processes and redirecting their attention to more important or more urgent environmental conditions, and
- reasoning efficiently and effectively about temporal processes, including the reasoning process itself.

1.2 Pre-Workshop Activities

In the first phase of this initiative, the Air Force provided seed funds for three teams, from Stanford University, University of Massachusetts, and Teleos Research, to begin laying out the issues and to help guide subsequent directions. The Air Force also chartered Cimflex Teknowledge to coordinate this effort and to organize a workshop to expose these results to a wider community and garner feedback. The overall goal was to provide a clear direction for subsequent research.

The individuals involved had all been working on IRTPS-related projects of various kinds. Now these teams, working independently and together over a period of about four months, studied IRTPS as a research domain and established preliminary ideas and answers to questions in several areas, including:

Terms and Issues:

> What technical questions should be included in IRTPS and, conversely, which questions should be excluded from consideration? What kinds of issues should be focused upon?

Experimental Methodology:

How should experiments be conducted and evaluated?

Common Experimental Problem:

What would be a pragmatically viable and effective common testbed for the IRTPS research community?

At the same time, the plans for the workshop were clarified: It would be two days, with the first day devoted to presenting and discussing the ideas and positions of the funded teams, and the second day spent in small working groups on particular topics. The workshop would be limited to a maximum of fifty attendees, and would cover as broad a range as possible of the experienced IRTPS community. Those applying to attend would submit an abstract describing their relevant experience.

The pre-workshop teams produced six reports. These were distributed to all workshop participants beforehand. Two of the attendee abstracts provided particularly astute overviews and were also distributed. An additional relevant paper was also distributed. Those nine reports (or, in some cases, revisions of them) appear as Appendices II to X to this document. Section 1.3 provides an overview of these pre-workshop papers.[1]

1.3 Overview of the Pre-Workshop Papers

Stan Rosenschein, Barbara Hayes-Roth, and Lee Erman formed an *ad hoc* working group and produced the paper "Notes on Methodologies for Evaluating IRTPS Systems" (Appendix II). They present a model of embedded dynamic systems and introduced neutral terminolgy. In particular, they separate the system, ***S***, from the environment, ***E***, but allow the ***S-E*** boundary to vary to allow investigating various kinds of IRTPSs. One measures a system by defining functions on time sequences of physical states; one also designates a subset of the measurements as utilities for evaluating the behavior of the ***S***. This approach, and especially the ***S-E*** terminology, was adopted throughout the workshop and provided useful commonality of terminology and concepts.

In "Report on Issues, Testbed, and Methodology for the IRTPS Research Program" (Appendix III), Yoav Shoham and Barbara Hayes-Roth, the Stanford team, expand on the model of Rosenschein, Hayes-Roth and Erman, modeling the system as a time-delayed automaton wired to and interacting with its environment -- an *informatic pair*. They provide definitions of some usually intuitive concepts, such as *timeliness*, *recency*, *unpredictability*, *uncertainty*, and *robustness*, in terms of the environment-machine model. They include a discussion and set of recommendations on testbeds for IRTPS, with emphasis on a variety of applications. They also discuss issues of real-time intelligent agents, and expand on those issues individually: Shoham's work (Appendix VI) emphasizes a formal model of an agent, its knowledge, and its interaction with other agents -- an approach he calls *agent-oriented programming*. Hayes-Roth's research program (Appendix VII), which she characterizes as aimed at *adaptive intelligent systems*, concentrates on real-time requirements and heuristics, architectures, and experimental evolution.

---

[1]This report includes several other sections. Appendix I provides brief descriptions of a number of systems and architectures being used for IRTPS-related work. Appendix XI provides post-workshop thoughts from two of the participants. Finally, the annotated bibliography is a compilation of items submitted after the workshop by various participants.

Tom Dean's abstract "Intelligent Real-Time Problem Solving Workshop" (Appendix IV) provides a neat description of his IRTPS perspective in relation to research on planning. In "Real-Time Problem Solving: Preliminary Thoughts" (Appendix V), N.S. Sridharan and Raj Dodhiawala also present a general overview of IRTPS, but from an engineering and architectural perspective.

In their "IRTPS Workshop Interim Team Report" (Appendix VIII), the Teleos-organized team of Stan Rosenschein, Michael Fehling, Matt Ginsberg, Eric Horvitz, and Bruce D'Ambrosio lay out the IRTPS problem. They break down the research programmatic issues into historical/ interdisciplinary, research on resource-bounded reasoning, and experimental validation. They discuss each of these succinctly.

Paul Cohen, Adele Howe, and David Hart present the University of Massachusetts team report in "Intelligent Real-Time Problem Solving: Issues and Examples" (Appendix IX) -- a revised and augmented version of their pre-workshop paper. Based on their ongoing research on a fire-fighting simulation called Phoenix, they offer a comprehensive vision, starting with their "behavioral ecology" view. This view is based on three interacting components: the agent's architecture and knowledge, the agent's behavior, and the environment structure and dynamics. They compare and contrast this to the ***S-E*** model presented in Appendix II and argue that many issues can be handled only by understanding the agent's architecture -- a point compatible with the *informatic pair* of Shoham and Hayes-Roth (Appendix III). They present their Phoenix agent architecture. They introduce the notion of "envelopes" -- spaces through which agents move that can be abstracted and used for plan development, monitoring, and agent performance evaluation. They also describe the Phoenix testbed.

In Appendix X, Ed Durfee and Thomas Montgomery describe the testbed they are building at the University of Michigan in "MICE: A Flexible Testbed for Intelligent Coordination Experiments." Although developed in the context of distributed AI, the testbed is relevant for IRTPS. The Phoenix and MICE testbeds provided the workshop testbed working group two concrete examples for description and analysis. (See Section 5.)

1.4 The Workshop

The workshop was scheduled in a somewhat remote location, to encourage and enhance a concentrated group experience.[1] After some introductions by Abe Waksman (AFOSR) and Lee Erman (Cimflex Teknowledge), the workshop opened with a short overview of application requirements. The person scheduled to make this presentation had to cancel at the last minute, and Armen Gabrielian graciously and bravely stepped in with less than an hour's notice for that.

---

[1]The workshop was scheduled to be held at the Inn at Pasatiempo, a motel in the mountains on route 17 northeast of Santa Cruz, California. On October 17, less than three weeks before the workshop, the San Fransisco Bay area was rocked by a 7.1 earthquake (the "Pretty Big One"), with the epicenter just a few miles south of Santa Cruz. Although the Inn suffered only superficial damage, access to the area was severely restricted because of major landslides and damage to route 17. With some trepidation about access problems, and a great deal of real-time problem-solving consideration, we stayed with the planned location. Fortunately, most people did not have undue difficulties. We did experience two consequences of the quake: The facilities were crowded because of the many insurance agents handling claims in the area, forcing some participants to stay at nearby motels, and we experienced several after-shocks, including a severe jolt during one of our sessions.

Team		
UMass:	Presenter:	Paul Cohen
	Discussants:	Ed Durfee, Les Gasser
Teleos:	Presenter:	Stan Rosenschein
	Discussants:	Stuart Russell, Kurt Konolige
Stanford:	Presenters:	Yoav Shoham, Barbara Hayes-Roth
	Discussants:	Tom Dean, Ted Linden
general discussion:	N. S. Sridharan	

Table 1-1: Team presenters and discussants

Each of the three teams presented a brief overview of their position. All participants had been requested to read the reports before the workshop, and the workshop agenda was predicated on that, avoiding the necessity to spend time presenting those materials. Thus, in these 45-minute presentations, each team could concentrate on an overview, emphasizing the parts they felt most important. Following each team presentation, two pre-assigned discussants took ten minutes each to respond to the team's position or otherwise present their views on IRTPS. Subsequently, they led the workshop attendees in 25 minutes of open discussion. (Actually, there was much interactive discussion during *all* segments of the workshop.) The presenters and discussants are listed in Table 1-1.

The participants split into four working groups for the second day, as shown in Table 1-2. The working groups met for two and one-half hours in the morning followed by 15-minute presentations each to all the workshop attendees. The groups met again in the afternoon, for about one and one-half hours, and then each presented their results for 15 minutes. Following the workshop, the group leaders wrote up the results of those meetings; those write-ups make up Sections 2 through 5 of this report. These reports attempt to capture both the substance of the discussions and the inclinations of the participants, including the reactions to the pre-workshop reports.

The workshop concluded with remarks by Nort Fowler (RADC). Nort emphasized that although the workshop and the proposed initiative are sponsored by the Air Force, that it is "your workshop, and your science," and that the participants and other researchers are ultimately responsible for setting fruitful directions.

1.5 Comments

Based on informal comments from many of the participants, the workshop proved very worthwhile. Distributing the preliminary reports ahead of time and strongly encouraging the participants to read them had the intended effects of providing a common base and drastically reducing the time spent on formal and background presentations. We also succeeded in gathering a diverse group of people; one participant noted and all agreed that it was a collection of researchers who rarely talk to one another. Yet there was continuous, intense, and highly interactive discussion among all the attendees.

Terminology and Criteria	
Planned Output:	A description/definition of an IRTPS system and a listing of criteria/utilities along which IRTPS systems should be measured. Examples of criteria include responsiveness, interruptibility, and graceful degradation.
Leaders/Editors:	Barbara Hayes-Roth and Yoav Shoham
Participants:	Thomas Dean, Tony Fountain, Dan Miranker, N.S. Sridharan, Mike Wellman, Bob Whitehair
Research Areas	
Planned Output:	A listing of research areas important within IRTPS. For each area, a list of questions, annotated with estimates for value and difficulty, and a research agenda of questions. Also, a listing of related issues and areas that should *not* be addressed in a near-term IRTPS research program.
Leaders/Editors:	Michael Fehling, Matt Ginsberg, and Stan Rosenschein
Participants:	Franz Barachini, Mark Drummond, Bob Filman, Armen Gabrielian, Don Geddis, Pete Halverson, Tomasz Imielinksi, Leslie Kaelbling, Kurt Konolige, Donald McKay, Stuart Russell, Marcel Schoppers, Josh Tenenberg, John Yen
Architectures	
Planned Output:	Descriptive characteristics which differentiate various IRTPS architectures. Examples of existing or proposed architectures which exhibit the various characteristics.
Leaders/Editors:	Paul Cohen and Paul Rosenbloom
Participants:	Raj Dodhiawala, Bill Erikson, Les Gasser, Ted Linden, Henry Mendenhall
Testbeds and Common Experimental Problem	
Planned Output:	Desirable characteristics for an IRTPS testbed and for a common experimental problem. Analysis of Phoenix (U. Mass.) and MICE (U. Michigan) as candidates.
Leaders/Editors:	Bruce D'Ambrosio and Lee Erman
Participants:	Bradley Allen, Bill Baker, Ed Durfee, Adele Howe, John Jensen, Perry McCarty, David Toms

Table 1-2: Working-group organization for day two

We had hoped to produce more concrete proposals for a research program. In restrospect, that may have been overly optimistic, primarily because of the relative youth of the topic and also because it appears to be quite broad. Most participants were loathe to restrict the range of research efforts of the field, even if they could dictate those restrictions.

2. Terms and Operational Criteria

Leaders/editors: Barbara Hayes-Roth, Yoav Shoham
Participants: Thomas Dean, Tony Fountain, Dan Miranker, N.S. Sridharan, Mike Wellman, Bob Whitehair

The goal of this working group was to identify, define operationally, and provide examples of key terms and evaluation criteria for intelligent real-time problem-solving systems.

We defined an intelligent real time problem-solving system (IRTPS) as: *an intelligent system that is sensitive to real-time constraints on the utility of its behavior.* By the term "intelligent," we imply the relevance of concepts of knowledge, beliefs, goals, capabilities, reasoning, and so forth to describe and explain the system's behavior (see [Shoham & Hayes-Roth 90] (Appendix III here). By the phrase "real-time constraints," we refer to the impact of temporal factors on the value of behavior that is otherwise correct, particularly where those temporal factors are determined by dynamic processes outside of the system's control. We assume that a system constrained to real-time performance may have to behave differently than one not so constrained and, conversely, that the system can use its intelligence in order to adapt to its real-time environment (see [Dean 90] (Appendix IV here), and [Hayes-Roth 90a] (Appendix VII here)).

2.1 The *S-E* Framework

To define performance criteria for IRTPSs, we adopted the neutral language of the ***S-E*** *framework* proposed by [Rosenschein, Hayes-Roth & Erman 90] (Appendix II here) (see also [Shoham & Hayes-Roth 90], Appendix III here). Briefly, the overall system is modeled as an embedded dynamic system and described as a time series of physical states, with mappings from instants of time to a state space of values that model the physical features of interest. Events are defined as changes in state values. Measurements on the system are defined as functions on state values or events, or series of state values or events. The overall system is partitioned into sub-systems corresponding to the IRTPS, ***S***, and the environment, ***E***, each of which has a dynamic local state that varies, in part, as a function of signals received from the other. The placement of the ***S-E*** boundary is variable to allow investigation of IRTPSs of variable scope, for example a "complete intelligent agent" versus an intelligent perception system. However, placement of the boundary determines the distinctive roles of particular measurements within the investigation. Measurements on ***E*** can be functions on internally generated features of the environment or features of the environment influenced by ***S***. The former can be used to assess the degree of difficulty or other qualitative features of the task facing an IRTPS. The latter can be used to assess the degree of an IRTPS's success or other qualitative features of its behavior in the environment. These measurements assume the introduction of *utilities* into otherwise descriptive functions. Measurements on ***S*** also can be functions on independently generated features of the IRTPS or features influenced by ***E***. The former can be used to explain how it is that the IRTPS produced the observed behavior. The latter can be used to explain its sensitivity to particular kinds of environmental variability. Although measurements on both ***S*** and ***E*** are necessary for a thorough investigation, we focused on measurements on ***E*** during the limited time available for group discussion.

We assumed that the ultimate performance criterion for an IRTPS is *to maximize a*

comprehensive utility function over time. Presumably, this utility function could be defined in the terms of the above framework. Because this function is likely to vary across applications or even across different contexts within an application, we did not attempt to define a global utility function. However, we assumed that there exist a number of subordinate criteria that, if met, would promote global utility maximization in a variety of circumstances. As candidate subordinate criteria, we reviewed the criteria defined informally in [Hayes-Roth 90a] (Appendix VII here): asynchrony, timeliness, selectivity, coherence, flexibility, responsivity, robustness (see also [Sridharan & Dodhiawala 90] (Appendix V here). The remainder of this review gives our definitions of those criteria we considered most important. (See also a similar effort to operationalize performance criteria in [Shoham & Hayes-Roth 90] (Appendix III here) and [Hayes-Roth 90a] (Appendix VII here).)

2.2 Specific Criteria

We defined *timeliness*, the minimal performance criterion, in terms of event-response pairs. Following the literature, we defined *hard* real-time constraints as restrictions on the time interval during which a logically correct response would have the desired effects. Hard constraints can be absolute, for example: "Save your files before the system is brought down at 5:00pm." Or they can be relative, for example: "Prepare your weapons within thirty seconds of detecting an unknown vehicle." By contrast, *soft* real-time constraints restrict the time intervals during which a logically correct response would have different degrees of desired effects, for example: "The sooner you finish this paper the better." In both cases, it is not when the agent initiates an action that counts, but when those actions produce their results in the environment. In the terms of our framework, hard and soft real-time constraints can be expressed as utility measurements on the occurrence of particular ***S***-response events in ***E*** during particular absolute or relative time periods.

Several of the other criteria discussed below seem to be special cases of timeliness or heuristic approaches to achieving timeliness.

Defining some of the other criteria requires a characterization of important features of the environment. We identified several important features intrinsic to ***E***: *frequency*, *diversity*, *predictability*, and *periodicity*. Each of these can be expressed as measurements on events in ***E***. For example:

- "Blood gases are measured once every hour."
- "A cold post-operative patient will warm to normal body temperature in about eight hours."
- "Each of one hundred physiological variables are measures once per second."

We also identified some features related to the interaction between ***S*** and ***E***. For example, we defined *overload* as a condition in which the rate of events produced in ***E*** and sampled by ***S*** exceeds the rate at which ***S*** can produce responses. We believe a taxonomy of features should be developed.

We defined *selectivity* as a timely response to an appropriate subset of events in ***E*** under conditions of overload. Other things being equal, the probability of a timely response to an event should be positively correlated with the importance of the event, where importance is an attribute assigned within an application. For example, while an intensive-care patient monitoring system

is in the process of predicting the potential consequences of a minor irregularity in the patient's condition, it should give minimal attention to other sensed patient data that are in expected ranges. However, it should immediately redirect all of its attention to diagnosing and correcting any critical (e.g., life-threatening) problems that occur even if doing so interrupts its ongoing prediction activities.

We defined *coherence* as a condition in which a sequence of responses by *S* over time matches some criterial pattern. For example, an intensive-care patient monitoring system would be said to be coherent if it completed those diagnosis, prediction, and other tasks it began, within a specified period of time, with minimal distortions resulting from extraneous activity.

We defined *flexibility* as a condition in which *S* could interrupt a sequence of responses in order to react appropriately to important non-task-relevant responses. For example, a patient monitoring system would be said to be flexible if it interrupted an ongoing prediction task to respond to a critical patient condition.

We defined *robustness* as the insensitivity of a real-time performance measure (e.g., timeliness and selectivity) to variations in some environmental attribute (e.g., event density and diversity). For example, a patient monitor would be said to be robust if it maintained a criterial level of timeliness on critical events over broad variations in non-critical event frequency. There exists a family of robustness measures corresponding to each combination of performance and environmental parameters.

Although the working group did not have time to consider all potentially interesting terms and criteria or to consider those above in the depth we would have preferred, we felt that the exercise had been productive in two ways. First, it produced an initial set of definitions of key terms. More importantly, it confirmed the feasibility of developing operational definitions of key terms within a neutral framework. Because this is a critical foundation for any comparison of alternative approaches to intelligent real time problem-solving systems, we recommend more systematic efforts to define a comprehensive set of terms and criteria.

3. Research Issues

Leaders/editors: Michael Fehling, Matt Ginsberg, Stan Rosenschein
Participants: Franz Barachini, Mark Drummond, Bob Filman, Armen Gabrielian, Don Geddis, Pete Halverson, Tomasz Imielinksi, Leslie Kaelbling, Kurt Konolige, Donald McKay, Stuart Russell, Marcel Schoppers, Josh Tenenberg, John Yen

3.1 Introduction

The working group on research issues was given the following set of tasks:

1. Propose IRTPS-related research areas.
2. Consider specific research topics relevant to IRTPS within each area.
3. Estimate each topic's potential value to IRTPS and its difficulty.
4. Identify research areas and topics *less* suitable for important, near-term IRTPS research.

Our deliberation was to include non-AI research areas, and we were asked to list research areas in other discipline. both within and outside of computer science) that are important to IRTPS and suggest the most important IRTPS-related research topics within these non-AI areas.

We began by considering the following four-stage model of how research on a major problem like IRTPS might progress:

1. Paradigms (initially termed "models")
2. Theoretical questions
3. Algorithm development
4. Technology base (tools and design principles)

Although the discussants generally felt that all of these stages are important to IRTPS, the discussion focu. d primarily on the first two and the relationships between them.

The discussion of this model of scientific progress produced the following:

- Instead of "paradigms" the term "models" was originally used for the first stage. However, subsequent discussion revealed that we had a more general range of activities in mind than the descriptive activity of modeling. Paradigms entail broad commitments to (a) prioritization of issues to be addressed, (b) a general vocabulary and set of constraints on its use in describing the phenomena of interest, and (c) preferred methodology for assessing alternative descriptions (predictions) and techniques produced within (i.e., consistent with) the paradigm. Therefore, commitment to a paradigm constrains its user both conceptually and methodologically in explorations of some set of phenomena. Initial activity at this stage centers around very informal distinctions and constructs. For quite some time there may be many, equally plausible, candidate paradigms. As a field matures the predominant paradigms are likely to yield formalized theory languages within which detailed models can be specified. Prioritization of research issues is significantly sharpened, and the methodology for measuring progress is also formalized.
- The role of "theoretical questions" was less clear. Some participants emphasized

how paradigms enable the formulation and examination of theoretical questions. In this sense, maturation of a paradigm logically precedes the theoretical examination used to refine that paradigm. Other participants emphasized the many important theoretical questions that must be asked and answered in order to formulate a paradigm in the first place. Such theoretical questions may be "pre-paradigmatic," or even "non-paradigmatic." In this alternative sense, examination of certain theoretical questions logically precedes articulation of a paradigm.

- Both of these views are valid. They provide complementary accounts of the relationship of theory development to analysis and experimentation. By combining these accounts one comes to see scientific progress as a complex, evolutionary process rather than a simple sequential ordering of stages.
- The theoretical questions that may be asked at the pre-paradigmatic stage are more likely to focus on broad issues and to be only informally stated. However, these informal questions play a very important role in the early evolution of a scientific discipline by helping to articulate and prioritize the concepts and methodology on which some more formal paradigm will stand.
- For computationally-oriented topics such as IRTPS, algorithm development (stage 3) and technology creation (stage 4) play an essential role as conclusive analytic and empirical tests, respectively, of the concepts and methods produced by a paradigm.

This broad discussion of scientific progress provided us a useful context within which to focus more specifically on the IRTPS initiative:

- IRTPS research is at an early stage of evolution. Consequently, it is important to support careful, though informal, examination of concepts and issues that may point the way toward choosing among competing paradigms or creating a new, more appropriate paradigm. There are many candidates for the role as a dominant IRTPS paradigm. These need to be better understood and evaluated.
- IRTPS researchers would benefit greatly from the existence of "paradigm problems" -- simplified problem instances whose features exemplify critical and broadly occurring features of the full range of IRTPS problems. For example, the well-known Sussman anomaly within the MIT "blocks worlds" serves as a paradigm problem that has stimulated much work within AI on reasoning about action. IRTPS needs its own stock of paradigm problems. However, it is not likely that one may successfully set out to formulate a paradigm problem. A problem achieves this status only if it becomes a widely studied problem, whose interpretation is generally agreed to.

In keeping with our discussion of the evolution of research, we attempted to generate plausible paradigms (models) around which IRTPS research could be organized. The next section presents the results of this analysis. We then exploited our knowledge of these paradigms to generate candidate research areas upon which effort might be focused in the IRTPS initiative, to refine topics within these these research areas, and evaluate their significance for IRTPS. Section 3.3 summarizes that discussion.

3.2 Candidate IRTPS Paradigms

Challenging concepts underlie the idea of an IRTPS system. Scientists are far from agreeing on the meaning of the concept of an "intelligent system" (whether computational or not). Nor are they ready to commit to a common understanding of what "problem solving" entails. Even the term "real-time" tends to generate vigorous debate. A number of disciplines suggest alternative perspectives and approaches to the development of IRTPS systems. Some of the more prominent candidates include the following:

Planning

This paradigm represents the AI mainstream. Problem solving is conceived as goal-oriented reasoning about action, planning, and "meta-planning." Advocates of this paradigm expect that further research will enable them to extend its basic concepts and techniques so that these methods exhibit real-time performance.

Logics

This paradigm is popular at Stanford's Center for the Study of Language and Information, among other places. It proposes to use special extensions of classical first-order predicate calculus to formally describe problem-solving, and now IRTPS, in terms of the interactions of an agent's beliefs, desires, and intentions. One set of logic-based approaches involves extending classical logic with modal operators, axioms, and inference rules intended to capture the "logic" of reasoning with beliefs, desires, etc. Another approach involves adding mechanisms for so-called "non-monotonic reasoning" that allow inference with assertions that may be retracted under some circumstances.

Algebraic Techniques

This paradigm was suggested by one participant as an alternative to logic-based approaches. The same goals hold but, in this paradigm, one uses and extends such formal constructs as universal algebra and similar formal systems. These algebraic techniques, rather than predicate logic, provide the underlying formal system.

Decision Theory

This paradigm views problem-solving as choice among alternative actions on the basis of preference. In this view, problem-solving is represented as choosing those actions that maximize a problem-solver's subjective utility. It puts forward the principle of maximizing (subjective) expected utility as a "gold standard" for chararcterizing ideal rational action. Advocates of this paradigm suggest that real-time problem-solving involves making rational choices while also taking into account the costs of thinking (i.e., computation) and of missing deadlines. IRTPS is thus conceived of as entailing a process to manage scarce problem-solving *resources* such as time and information. Decision theory is sufficiently mature as a paradigm to have well-tested mathematical methods for explicating its view. The formal methods of decision theory enable the explicit modeling of the problem-solver's probabilistic uncertainty, its preferences, and its attitude towards risk.

Control Theory

This paradigm has culminated in a mature academic discipline that has produced a large amount of practical technology for building real-time systems. Modern control systems manage complex distributed physical processes, and often do so while meeting real-time performance constraints. Control theory has also produced practical technology for building simple learning systems. Modern control theorists often conceive of an advanced control system as a non-trivial embodiment of intelligent (or at least intelligently created) problem-solving processes. Control theory and decision theory are closely related. Indeed, the theoretical language of decision theory can be used to describe many of control theory's concepts and formal constructs. However these paradigms differ in some important ways. In particular, applications of control-theoretic methods often model a "closed loop" relationship between the actions of a controlling system and the controlled system that it manages. By contrast, the formal methods of decision theory make it difficult to model feedback. Secondly, advanced control theories have been developed that emphasize specifying a control response by analyzing the feedback and other observations of the "plant" (the controlled system) in terms of a reference model available to the controlling system. Model-reference methods can emphasize qualitative issues that have received less attention in decision theory. Examples include use of a reference model that only approximately models the actual dynamics of the plant and methods for replacing a state-based model with one that has fewer states but which can still be used to meet the original control criterion (i.e., so-called state-aggregation methods). In sum, although control theory and decision theory are closely related paradigms, they provide significantly different emphases and may even turn out to be formally incompatible.

State-Based Automata Methods

One discussant proposed state-based automata methods as an alternative paradigm. The principal distinction of this paradigm seems to lie in its use of the mathematics of automata theory for formal modeling and analysis. There may also be some increased emphasis upon the control of non-linear systems in this approach, although this difference from standard control theory has diminished in recent years. In fact, many state-based methods are well-known to control theorists, and many of the standard problems defined within the state-based paradigm are similar or identical to the standard problems emphasized by control theorists, e.g., reachability, identifiability, stability, convergence, and controllability.

No-Paradigm

Several of the participants pointed out that IRTPS (and AI in general) is arguably at a very early, pre-paradigmatic stage. No existing paradigm obviously provides explicit coverage of all IRTPS issues. In particular, no existing approach can readily claim to have been formulated explicitly with IRTPS in mind. So, we should not attach ourselves to one of these existing paradigms with the illusion that it provides a suitable basis for investigating IRTPS. Given that this is the case, there are at least three alternative courses of action:

- Do something to develop a new paradigm from scratch. Unfortunately, it is difficult to imagine how to guide this process, or even to suggest how work of this type could be evaluated or supported. Most likely, one can judge the value of a new paradigm only in retrospect.

- Compare alternative existing paradigms and then adopt the best ideas from the closest or some combination. This variant probably describes the approach of many AI researchers in the history of that field. The paradigm labeled "Logics" above may exemplify this to a certain extent.
- Look for some deeper, more foundational paradigm within which IRTPS is a "special case." This alternative is strongly reductionist and presumes that one has a coherent sense of the dimension of reduction and a convincing argument for the acceptability of the chosen paradigm for the underlying level. As an example, some AI researchers have attempted to rely upon work at the foundation of mathematics as a deeper conceptual and formal basis for their approach to AI problems. Prima facie, this strategy may be problematic for IRTPS because of foundational difficulties in modeling notions of *process* and *time*.

3.3 Toward an IRTPS Research Agenda

Our discussion group contained advocates of all the candidate paradigms discussed above. We exploited these diverse perspectives in generating a set of candidate IRTPS research areas.

After some discussion and re-organization, we agreed on a rough outline of the areas and sub-areas within which we expect to find research problems critical to IRTPS. We distinguished these areas as being either issues of resource-constrained reasoning or problems of modeling a real-time environment. The subsequent discussion of topics focused on only the former (with some rearrangement.)

The following outlines critical IRTPS-related research areas:

1. Problem-solving under resource constraints
    a. Limited resource reasoning
        i. Controlling focus of attention
        ii. Hierarchy of reflection
        iii. Temporal reasoning
        iv. "Anytime" reasoning
    b. Managing varying resources at "runtime"
    c. Managing competing objectives
    d. Reasoning about resources
2. Modeling features of a complex, dynamic environment
    a. Uncertainty and unpredictability
    b. Time-dependent events and action outcomes

After settling on this preliminary categorization of IRTPS research issues, our afternoon session focused on two subsidiary goals: First, we attempted to identify fairly specific research questions within each of the broad areas that had been discussed in the morning and second, we attempted to evaluate each of these research questions with regard to three properties:

1. How likely was it that progress would be made in this specific area in the next two years?

2. How important to the IRTPS venture would this progress be?
3. How "easy" was the problem? This term was not defined further, but seemed to measure the group's collective confidence that progress predicted in (1) would in fact be achieved.

Of course, all of the decisions made about these questions were entirely subjective and should be taken principally as reflecting the opinions and biases of the research issues working group.

3.3.1 Short-Term, Important Issues

The following research problems were felt to be important ones on which progress could be expected in the near term:

1. Compilation. The problems here are those of preprocessing information so that it can be accessed more quickly when it is needed, or transforming one representation of a process to another, more efficient, representation. The techniques mentioned included inductive methods, the synthesis of causal theories, case-based reasoning, and neural networks. This is such an active research area already that it was felt certain that progress would indeed be made over the next two years, and compilation was therefore labeled "easy."
2. Resource representation and monitoring. What problems are involved in representing the fact that IRTPS systems can be expected to encounter resource limitations? What needs to be done to make it practical for these systems to monitor their resource consumption and needs? The first of these was felt to be easy; the second, less so.
3. Time-dependent utility. How can an agent expect the utility of achieving certain goals to vary over time? Not easy.
4. Metareasoning applied to planning and to search. Applying some sort of metalevel reasoning to control search, to determine when replanning is needed, or to weigh competing subgoals. Easy.
5. Sensory planning. Planning to acquire new information through the use of sensors. Easy.
6. Plan monitoring. Given that a plan has been constructed to achieve some real-time goal, how can progress be monitored as the plan is executed? Not easy.
7. Anytime reasoning. Here, we identified a variety of subproblems that could be lumped under the title of "anytime inference:"
    a. Redefining inference in a way that allows the inference process to be interrupted, and constructing algorithms that capture this new definition.
    b. Constructing measures on the value of an incomplete answer to a query.
    c. Defining a notion of an "approximate" answer to a query. (Not short-term.)
    d. Progressive planning, by which we mean the development of planning systems that work by progressively improving/refining partial plans.

    None of these problems was felt to be easy.

8. The application of decision theory to the problem of focusing attention. What should the agent concentrate on next? Which of its sensors should be polled at any particular time? Not easy.
9. Finally, we included in this group of problems that of investigating the foundations of intelligent, real-time problem solving. It was felt that progress would inevitably be made here.

3.3.2 Important Issues, But Not Short-Term

These problems were generally felt to be harder than those in the previous section:

1. Resource planning. Planning to obtain more resources.
2. Process representation. Understanding and representing the time-dependent processes with which IRTPS systems will need to interact.
3. Utility of metareasoning and learning. Given that we reason about our own problem-solving activity and that we learn, under what circumstances are these mental activities useful ones?
4. Understanding and being able to deal with competing goals.
5. Understanding dynamic focus of attention. How should an agent's focus of attention change from time to time?
6. Understanding partial or incomplete plans; debugging incorrect plans.

3.3.3 Not Currently Viewed as Important

Finally, several research areas were identified that were felt to be of lesser importance to the IRTPS effort. In some cases, this was because these problems were not felt to lie within the IRTPS domain specifically; in others, the problems simply failed to generate any enthusiasm within the research issues working group.

1. Learning metalevel information by induction.
2. Caching. By this we mean the study of data structures and mechanisms that could be used to save previously computed results.
3. Metalevel reasoning applied to scheduling, to situation understanding, and to probabilistic computation.
4. The application of operating systems concepts (interrupts, priorities, etc.) to the problem of focusing attention.
5. Defining the concept of a plan.
6. Planning for goals that are partially satisfiable or temporally scoped (such as that of maintaining some external condition). This and the previous topic were felt to be the domain of the planning community proper and outside the scope of IRTPS per se.

3.4 Concluding Remarks

The working group was aware that the background and interests of its members may not have been fully representative of the IRTPS community as a whole and may have introduced certain biases into its conclusions. For example, the lack of representation of more researchers with strong applications experience probably limited our ability to generate a comprehensive list of the "right" questions. We were also restricted in our ability to propose paradigms because important, non-AI disciplines were under-represented. The "AI planning" contingent, on the other hand, was proportionately over-represented, and as a result, our discussion of research topics tended to be biased towards issues within that paradigm. These observations about the composition of the group are offered not to diminish the validity of our conclusions, but rather to place them in their appropriate context.

4. Architectures

Leaders/editors: Paul R. Cohen, Paul Rosenbloom
Participants: Raj Dodhiawala, Bill Erikson, Les Gasser
Ted Linden, Henry Mendenhall

4.1 Introduction

The working group began with a summary by Paul Rosenbloom of some prior discussions with Cohen and Gasser, and with three discussion questions:

1. What is an AI architecture, and specifically, an IRTPS architecture (hereafter *AIRTPS*)?
2. What should be the output of research on AIRTPS?
3. Which aspects of AIRTPS need more work/are most ripe?

Each was discussed at some length, but with different degrees of consensus.

4.2 AI Architectures

We started with a discussion of what a computational architecture is, and then attempted to specialize the definition to be an architecture for IRTPS. Rosenbloom offered four possible definitions or senses of architecture.

1. Architecture as programming language
2. Architecture as a theory or a model
3. Architecture as a commitment to a set of design decisions

The last one is close to what we want, but does not carry the idea that an architecture must be "complete" in some way. Thus, Rosenbloom proposed (and achieved consensus with):

4. An architecture is a "fixed structure," that provides a "flexible part" that can be programmed.

The fixed part is immutable (if you change it, you have a different architecture) and characteristic to the extent that it has been designed for particular kinds of tasks. (There was anticipated disagreement on the question of whether architectures are designed for particular tasks or are general to many tasks; see below.)

The flexible part is where task- and problem-specific knowledge resides. It is put there by programmers, knowledge engineers, and automatic knowledge acquisition or learning components. It includes both substantive knowledge (e.g., facts and heuristics) and control knowledge (e.g., skeletal plans and chunks).

By definition, the fixed part of an architecture is changed only as a last resort. If you want to change the behavior of an agent, you change its flexible part or construct it to change its own flexible part. The key question for an AIRTPS is what should be fixed and what should be flexible. If capabilities are in the fixed part, they are guaranteed---the architecture will always

behave that way. This is both an asset and a liability, because these capabilities are then difficult to modify. In contrast the flexible part allows arbitrary knowledge and methods to be used to enable and control behavior. This leads to an inherent difficulty in predicting behavior, but allows easy adaptation to new situations. The main time issue in the fixed part is time boundedness of the architectural components -- without the ability to use arbitrary knowledge, combinatorics in the the fixed part would kill you -- while the main time issue in the flexible part is the control of combinatorics. Real-time algorithms for different situations probably go in the flexible part; whereas mechanisms for handling communications, interrupts, and low-level scheduling probably go in the fixed part.

For another view of the question, "What goes in the fixed part," we asked what is the basic set of questions that, once answered, give the specifications for an architecture. One path to this is to start with a set of abstract criteria -- such as universality (in the Turing sense), autonomy, etc. -- that must be achieved by any architecture. A second path is to try to get a sense of what more detailed capabilities/mechanisms must be incorporated into a "complete" AI architecture: memories for the storage of knowledge; capabilities for adding new information to memory (learning) and accessing it in a timely way; perceptual and motor mechanisms for interacting with the environment; selective mechanisms for attention, decision making, etc.; mechanisms that enable goal-oriented behavior, exponential search, and planning (and the control of such behavior); and mechanisms that enable the use of language. It seems worthwhile to complete this list, especially in light of the list of requirements for AIRTPSs discussed below. This list switches back and forth between "mechanisms for..." and "mechanisms that enable..." because, in general, the architecture need not directly provide an entire capability. Minimally, it must provide a set of fixed mechanisms that can be completed by the addition of appropriate structures in the flexible part. How much of any capability is directly supported by the architecture, versus being defined as knowledge and programs in the flexible structure, is one of the major variations across architectures.

As architects, we should be designing the fixed parts of AIRTPSs, but, because of the ways capabilities can be split between the fixed and flexible parts, we cannot do that without thinking about the knowledge we expect to have in the flexible part. For example, we probably do not want to spend all our time designing anytime algorithms, but we do want to design architectures that will support anytime algorithms.

Having achieved a degree of consensus on the fixed/flexible view of architectures, we addressed a few remaining general issues in architecture design and then turned to questions that were specific to IRTPS. We noted that architectures are frequently layered, and identified three layering topologies:

1. **Layering by time constants (levels of implementation).** Rosenbloom noted (based on observations by Newell) that successive layers of SOAR have cycle times increasing by roughly one order of magnitude. In SOAR, this layering occurs because units of behavior at higher levels are composed from -- that is, implemented by -- multiple behaviors at the levels below, and thus take proportionately longer. Phoenix has a similar decomposition by time constants, though it is not viewed as being layered, because there is no comparable implementation relationship between them; Phoenix has two distinct action-generation components, one fast and reactive, and one slower and more contemplative. For more on layering, time constants, and the idea of time scales in the environment, see Section 4.4.

2. **Vertical decomposition.** In this, the most common layering, the layers contain modules that are successive in the direction of information flow from the environment.

3. **Horizontal decomposition.** At each level we find complete capabilities of increasing sophistication. The lowest levels contain the simplest sensory processing, motor schemes, reflexes, and so on. Higher levels contain more symbolic abilities, such as planning. Brooks, when he introduced this kind of decomposition, called it a subsumption architecture.

An AIRTPS must make some guarantees about timeliness in its fixed part. Only the layering by time constants addresses this explicitly, although we expect that the subsumption architecture probably has de facto layering by time constants. What is not clear, in the subsumption architecture, is which parts are fixed and which flexible. Whereas the lower levels of the vertical decomposition are more basic in the sense that higher levels depend on their outputs (and thus the timeliness guarantees), the lower levels of the subsumption architecture produce behavior that is gated, not consumed, by the higher levels. It may be that the timeliness of each level of competence in the subsumption architecture has to be guaranteed separately.

4.2.1 Architectures for IRTPS

The first question we considered was whether an AIRTPS needs to be a general AI architecture. This question had many interpretations, so the authors offer these post hoc clarifications and opinions.

Does an AIRTPS have to be general, in the sense of "built for lots of different environments," or specific to particular environments. Here we found no consensus. One view takes its inspiration from the array of environments in which humans can behave effectively. The view is that an appropriately constructed adaptive system can learn to behave well -- though not necessarily optimally -- in a wide variety of environments. The other view takes its inspiration from biology, where the environments in which humans, or any other animal, exist are just a fraction of the totality of all environments. The view is that some architectures are significantly better than others in particular environments. And while it is true that humans are marvelously well adapted to a remarkable range of environments, it is also true that humans are less well adapted to most of those environments than their other inhabitants. Thus, if we view the AI enterprise as building something like a human, then we try to build a general architecture. But if we view AI as design, then we want to know how the subtle differences in environments produce subtle differences in designs, and we eschew the search for a general design that does moderately well despite the differences.

Another interpretation of the question was, does an AIRTPS architecture have to be *complete* in the sense of displaying the range of behaviors we expect of intelligent agents, including perception, learning, planning, and so on. Here, we seemed to have consensus that an AIRTPS must support these activities, although there was some debate about whether learning had to take place in real time. An alternative was to have real-time performance but off-line learning.

Given what we agreed upon, that an AIRTPS should support many or all of the activities that we find in AI architectures, the next question concerned how we, as designers, should develop AIRTPSs. Four strategies were suggested:

1. Modify known AI architectures for RTPS.
2. Design AIRTPS architectures from scratch.
3. Modify known RT architectures (operating systems and languages) to be more intelligent.
4. Combine known AI and RT architectures.

The extreme interpretations of the first two approaches are ideological poles. During the Workshop we heard, on a several occasions, that you cannot modify AI architectures for real time by "adding a real-time box" (Sridharan's phrase). None of us (so far as we know) interpreted the first approach as simply adding a real-time box. Rather it would involve some mixture, as necessary, of modifying existing mechanisms -- for example, modifying SOAR's production matcher to eliminate the possibility of exponential match times -- and the addition of new mechanisms. For well-developed AI architectures that are not too far from the demands of real-time behavior, this can be a cost-effective strategy for creating an AIRTPS. The second approach, though apparently expensive, was the most attractive to some of us, and is not necessarily expensive in the long run. Those of us who view AI as design see, in real time environments, new constraints on design, and new opportunities to understand how the design is influenced by the environment. The third approach seemed promising and interesting, so we were surprised when the "Basic Research" working group panned it. Admittedly, understanding real-time operating systems is only scholarship, and applying RTOS concepts to AI is just engineering, but this humble approach actually has an advantage: If you do not insist on inventing real-time methods yourself, but are content to adopt and adapt them, then you can spend your time building real-time systems, and testing, analyzing, and explaining why the methods work or do not work in particular environments. The fourth approach attempts to build as little new structure as possible, by combining a bounded, but limited, real-time architecture with an unbounded, but general, AI architecture. Control would pass between the systems, as determined by the demands and resources that were available.

We left the general topic of AIRTPSs with a question that would concern us again later: How do we decide on a set of AIRTPS capabilities? Having voiced the question we turned to our second topic of the day.

4.3 Projected AIRTPS Research Outputs

Four products of AIRTPS seemed attainable and worthwhile:

1. Architectures for real-time problem solving
2. Evaluations of architectures for real-time problem solving
3. Methodologies for designing new AIRTPS, and for selecting among existing ones, for given environments
4. Programming methodologies for specific architectures

We achieved consensus that AIRTPS research would move faster if we had some AIRTPSs to work with, and, thus, their development was a priority. Beyond that, opinions about priorities diverged, although the disagreements were minor and most agreed that evaluation is important.

Les Gasser cautioned against extremism in evaluation, saying that in AI, concepts often turn out to be useful for unforeseen reasons. Someone else said that because architectures are supposed to cut across a lot of domains, it might be difficult to prescribe evaluation methods.

We began to enumerate evaluation criteria. These included the quality or performance of the AIRTPS, whether the AIRTPS addresses the key real-time issues, how easy is it to use, and whether it is an integrated architecture or a set of kludges. Many of the criteria mirrored the "quality attributes" discussed by Sridharan: modularity, extensibility, elegance, evaluability, real-time performance. All these seem innocuous but weak criteria, and the question of how to evaluate an architecture remains open. Moreover, the answers depend on how one views the AI enterprise, as we mentioned earlier. Architectures are, variously, theories or models of general cognitive mechanisms, designed artifacts for specific tasks and environments, and engineering tools for expediting system building. One's view of AI colors one's opinions about the role of architectures and, thus, how to evaluate them.

In sum, we agreed in general about the output of AIRTPS, but disagreed on the details. Rather than arguing about the details, we started a broad, nonideological search for research opportunities within IRTPS.

4.4 Research Needs and Opportunities

Like the "Basic Research" working group, we quickly generated a long list of research opportunities. Constrained (or rather, focused) by our emphasis on architectures. We got as far as the following incomplete list:

1. anytime algorithms
2. approximate processing
3. support for tracking the correspondence between processes (one of which is time)
4. support for rapid shifting of focus of attention
5. adaptation (short-term tuning)
6. support for computations that are known to be bounded in time, so you do not have to reason about the time a process takes (also called "design-time" engineering).

But at this point, methodological concerns took over. A word on methodology: Most of us would rather be doing AI research than thinking about *how* to do it (some participants said so explicitly), but lately we have needed to do both. Certainly, we could have generated a long, long list of AIRTPS research and engineering projects, but the sentiment seemed to be that completing the projects on the list would not necessarily guarantee that we understood real-time problem solving well enough to implement it in any environment. Someone asked whether we could exploit what we already know about AI architecture design, and this led us to ask whether we could avoid the mistakes that had characterized AI architecture research in the past. Cohen pressed his view that architecture design must be constrained by the environment---that different environments necessitate different architectures. Gasser or Rosenbloom (or perhaps both) said Cohen's was a coercive view, that an effective approach for designing general architectures was to start the architectural design process with generalized knowledge gleaned from past experience with a range of environments and mechanisms, and then to challenge the resulting architecture with new environments (leading possibly to tuning of the architecture).

With no fundamental resolution of the argument in sight, we decided to move on, and examine an environmentally-driven strategy, consisting of: (1) generating environmental characteristics critical to domains requiring IRTPS; (2) evaluating the state of the art, as expressed in existing architectures, with respect to handling these characteristics; and then (3) focusing on the characteristics where research effort would most productively be spent.

When we started generating critical environmental characteristics, something notable happened: The first two entries were "near-instantaneous response," and "intelligent reactivity---the system should be able to tune its reactions." Henry Mendenhall was the first to notice that these are not characteristics of environments at all, but are characteristics of architectures, or systems built within architectures. To at least some of us, this slip seemed paradigmatic: when asked to describe a problem, we instead describe the solution. Chastened, we set to the task of listing characteristics of environments that either defined or exacerbated real-time problems:

- lots of data
- low-signal-to noise ratio
- unpredictable rates at which data arrive (varying quantity of data)
- hard and soft deadlines
- time-dependent value of knowledge and actions
- spectrum of environmental predictability
- incompleteness in environmental characterization
- multiple time scales (orders of magnitude) at which responses are necessary
- combinatoric possibilities to filtered through

Most of these are self-explanatory. We did have a brief discussion of the idea of time scales. Cohen proposed it for the list but Gasser argued strongly that time scales cannot be defined independent of an agent, and thus scale is not strictly a characteristic of an environment. We agreed on this definition for time scale: the average time between causal event cycles that have value for the agent. Some cycles are very short (e.g., the time between moving into a fire and getting burned) and some are much longer (e.g., the time between a wind shift and the recognition of failure of a fire-fighting plan). We also suggested that, as a design strategy, it is sometimes better to have two or more parallel components for events that happen on very different time scales (similar to the idea of layering by time constants, above) than to have one component that must respond to events on all the time scales.

Upon examination, Gasser's observation that time scale is not independent of architecture applies to several other entries on the list, too. Consider predictability. When we say an event is unpredictable, do we mean unpredictable for *any* conceivable agent or unpredictable for particular agents? Typically, we mean the latter. Similarly, deadlines are not inherent in environments, but instead are interpretations, in terms of changes in value to the agent, of events (and coincidences of events) in an environment. Apparently, many "characteristics of environments" are not really inherent, but are instead shorthand descriptions of problems that particular agents face in environments.

Some people wondered why, if our concern was real time, we were bothering with issues such as signal-to-noise ratio, unpredictable amounts of data, uncertainty of various kinds, and other things that had no apparent relationship to IRTPS. Acknowledging this point, many of us felt that these problems were common in IRTPS tasks and should be included in the list.

Given the list of environmental characteristics, the next step was to list the existing architectures that seem to offer something to IRTPS. The ones we came up with were:

- SOAR [Laird, Newell & Rosenbloom 87, Laird, Rosenbloom & Newell 86] For architecture/system, see I.12.
- Phoenix [Cohen *et al* 89, Howe, Hart & Cohen 90] For architecture/system, see I.9.
- BB1 [Hayes-Roth 85, Hayes-Roth 90b]
- RT-1 [Dodhiawala *et al* 89] For architecture/system, see I.11.
- ALV [Payton 86]
- Subsumption Architecture [Brooks 86]
- GAPPS [Kaelbling 88] For architecture/system, see I.4.
- PRS [Georgeff & Lansky 87, Georgeff & Ingrand 89a] For architecture/system, see I.10.
- DVMT [Durfee & Lesser 88, Durfee 88, Lesser & Corkill 83] For architecture/system, see I.2.
- RUM [Bonissone 87a]
- G2 [Wolfe 87]
- ABE/RT [Lark *et al* 90] For architecture/system, see I.1.

We also made a list of additional architectures that were interesting and instructive, though their relationship to IRTPS was not yet as clear as for the ones in the first list:

- THEO [Mitchell *et al* 89]
- PRODIGY [Minton *et al* 87, Minton *et al* 89]
- ICARUS [Langley *et al* 89]
- CYC [Lenat, Prakash & Shepherd 86]
- ISIS [Fox & Smith 84]
- ACT* [Anderson 88]

We then began the process of trying to analyze the architectures, with respect to how they dealt with the environmental characteristics. We constructed a table for ABE/RT (which, for arbitrary reasons, was the first architecture in our list) and asked how it dealt with a subset of the environmental characteristics:

- exp -- combinatoric possibilities to be filtered through
- low s/n -- low signal-to-noise ratio
- incompletely char.env. -- the agent does not know all the relevant information about an environment
- ***S-E*** match (deadlines) -- the agent must monitor processes in the environment (including the passage of time) and coordinate its activities to those processes.
- multiple scales -- some events in the environment happen "quickly" and some "slowly" for the agent.

To each of these questions, we gave one of four answers, with two possible qualifications:

1. n -- the architecture does not support the capability.

2. f -- there is provision for the capability in the architecture (the fixed part). f+ means it exists and is implemented, f- means it is under design.
3. v -- there is provision in the knowledge and methods encoded on top of the architecture (the flexible part). v+ means it exists and is implemented (although knowledge engineering may be continuing); v- means it is under design.
4. f&v -- both the fixed and flexible parts are involved.

SOAR:		
exp	f+v+	f: makes decisions, generates subgoals, and learns new knowledge; v: contains knowledge of options (defines exponential), knowledge about what decision to make (controls exponential), and knowledge on which learning is based
low s/n	n	
incompletely char.env.	f+v+	uncertainty leads to subgoals in which processing (e.g., search) can be performed to reduce the uncertainty. Also acquires more accurate models of environment by learning from success and failure
S-E match		
deadlines	f-v-	f: working on bounding architectural components; v: working on time-sensitive problem solving methods
multiple scales	f+	f: Starting from the bottom there is recognition, decision making, cognitive steps, and goals. Recognition is quick and automatic (associative, reactive, reflexive, etc.) with speed decreasing, and deliberation increasing with height in the hierarchy.
Phoenix:		
exp	f+v+	skeletal planning to handle combinatorial spaces, approximate processing to do as much search as there is time
low s/n	n	
incompletely char.env.	f+v+	Error recovery, handled in the same way as other actions, changes plans and actions when the environment changes unexpectedly or is not as expected. Reflexes handle very short term adjustments.
S-E match		
deadlines	f+v+	Envelopes and the timeline handle coordination.
multiple scales	f+v+	Multiple components (cognitive and reflexive) for generating action at different time scales

Table 4-1: Characterization of SOAR and Phoenix, as examples

We did not try to evaluate the quality or extent of the solution provided by the architecture, only whether there was one, and what the specific mechanisms were that provided it. Though we started with ABE/RT, we did not know it as well as others of the architectures, so we soon moved on to SOAR and Phoenix, as described in Table 4-1.

Unfortunately, we ran out of time before we could continue the comparisons with other systems and determine which issues deserve the most research attention. However, the general strategy seems worth pursuing. When it became apparent near the end of the day that we would not be

able to examine more that a handful of architectures, we shifted to a strategy of listing major classes of mechanisms that appear in some form in multiple of the existing architectures and which appear to help with numerous of the critical environmental characteristics. The list is by no means exhaustive, given the time available, but it did contain:

Open decision cycle
: At some time scale there is a cyclic decision process that is potentially open to: (1) all of the system's goals; (2) all of the system's knowledge (including knowledge about what to do); and (3) the input from the environment. Such a decision process is what allows the systems to be goal-directed, knowledge-intensive, and reactive (to both time and environmental contingencies), as appropriate for the situation. This contrasts with more rigid approaches in which decisions are driven primarily by plans and/or programs, and which therefore have only limited ability to respond to run-time contingencies. There may be more than one of these decision cycles, especially for organizations of agents.

Error recovery
: Neither events nor the timing of events in the future are predictable, so AIRTPSs must make provisions for error recovery. It is not clear whether error recovery should be permitted if there are hard deadlines.

Prediction
: What will happen, and when it will happen. Although uncertain, predictions are still necessary in IRTPS if there is to be any advanced planning. Indeed, if we accept that the "real-time problem" is that value functions are not monotonic over time, then predicting the maxima of these functions is essential to successful IRTPS. Predictions can be made by projection or can be compiled into reflexes.

Reflexes
: These are relatively fast sense-act loops with bounded (and closed) computation. Given the need of an open decision cycle to bring to bear a range of knowledge and input, such deliberative decisions are inherently slower than fixed reactions. Reflexes enable system performance at time scales smaller than the time required to make an open decision. Reflexes may be engineered into a real-time system or acquired during problem solving, either from an expert (e.g., Gruber's ASK system) or learned, as in SOAR.

Compilation
: The idea here is to automatically replace unbounded computation with functionally equivalent bounded sense-act reflexes. Chunking in SOAR is an example. Any agent that interacts with its environment and *does not* do this is wasting an opportunity, so we would expect to see it in all AIRTPSs in future.

Control reasoning
: Ordinarily control means "doing the right thing," but in IRTPS it means "doing the right thing at the right time."

In addition, there was a brief discussion of meta-reasoning, but everyone seemed to mean something different by the term (i.e., we had a representative sample of AI researchers in the group).

4.5 Conclusion

The architecture working group discussed a wide range of issues. Some were very pragmatic; for example, where should we concentrate research effort in the near term, and what results can we expect. Others went to the heart of the AI enterprise and illuminated differences in our goals and methodologies. Although we did not agree on everything, we surely agreed that real-time problem solving is a *fundamental* problem for AI because it challenges us at every level: pragmatic, technical, theoretical and methodological. We also agreed that these discussions lay the groundwork for a more thorough study of the space of architectures for IRTPS, including in-depth studies of existing and novel architectures, and the relationship of these architectures to the key isssues and environmental considerations of IRTPS.

5. Testbeds

Leaders/editors: Bruce D'Ambrosio, Lee Erman
Participants: Bradley Allen, Bill Baker, Ed Durfee,
Adele Howe, John Jensen, Perry McCarty, David Toms

Our working group was charged with discussing and recommending the role of a "testbed" in IRTPS research. We began with the following questions:

1. What is a testbed?
2. Why would we want one?
3. How should we use one?
4. Is there one available which might suit our needs?

It rapidly became clear that the answers to these questions were interrelated. We all agreed that a testbed was some kind of software platform for performing controlled, repeatable, instrumented experiments in IRTPS. That is, we acknowledged the possibility of, but did not spend significant time discussing, either "softer" (e.g., a paper problem definition) or "harder" conceptions of a testbed (e.g., a physical testbed like the autonomous land vehicle).

Subsequent sections of this report will discuss each of the four items above in turn. In this discussion we adopt the ***S-E*** (system/environment) terminology of [Rosenschein, Hayes-Roth & Erman 90] (Appendix II here); this terminology is summarized in Section 2.1 of this report.

5.1 A Testbed Framework

We attempted to elucidate a framework for describing, constructing, and sharing experimental setup (testbed) components. We identified the following components of a testbed, listed in approximate order from lower-level to higher-level:

Platform — The computing environment, including hardware, operating system, and programming system. Presumably one would not produce a new platform for a testbed, but the availability and familiarity of particular ones surely influence the choice. In addition to the usual desires that accompany most experimental AI, IRTPS work has a unique need: the ability to monitor and manipulate finely the temporal aspects of process scheduling.

Utilities — A set of basic functions useful for constructing IRTPS testbeds. We identified at least three categories of these:

- Execution utilities -- These directly support the running of simulations. Most notable here is a discrete event simulator.
- Experimental interface utilities -- There are three interfaces of concern: experimenter-testbed interface, system-testbed interface, and environment-testbed interface. The experimenter interface utilities include facilities for initiating, controlling, and observing simulation runs. System and environment utilities include handling test cases, logging, display support, etc.
- Analysis utilities -- Post-run support utilities for producing summaries and analyses of logged data.

S-E language — This is a language for defining systems and environments. This language might exist as a paper specification, providing a way to describe and compare systems, or exist as an implementation capable of accepting such descriptions and executing them.

Problem — An environment and a set of system tasks within that environment. This might be either a paper description or an implementation. Sample scenarios or test cases are useful parts of this component.

Base System Knowledge — A description of the knowledge available to a system for accomplishing the tasks described in the problem statement. It is desirable that this be as high level as possible, so as not to prejudice system design.

Behavior-Metric Mapping — This is a definition of evaluation criteria for system performance. The behavior-metric mapping defines a relationship between system behaviors, describable in the *S-E* language, and utility.

5.2 Motivation

There are really two questions buried here. The first is whether or not, and if so why, we think an experimental methodology is appropriate. The second is whether and why we might want to share one or more testbed components among research groups.

There was general agreement that an experimental methodology was essential: The kinds of systems involved in IRTPS, and the measures on quality of result, are too complex to allow for analytic derivation of many of the interesting behavioral properties of whole systems. This immediately raises a question regarding the generalizability of experimental results. We believe that careful selection of an experimental domain, and careful experiment design, can yield results with broad applicability. Some dangers and safeguards are discussed in the section on experimental methodology (Section 5.3).

Second, once an experimental method is adopted, sharing seems important, although sharing might mean several things:

1. We need to share results, and this must include descriptions of the experimental setup under which the results were obtained.
2. Researchers new to the field may wish to use existing testbed components to avoid the expensive investment in constructing a custom experimental setup.
3. Researchers at diverse locations might wish to replicate experimental conditions varying only certain parameters in order to obtain comparable results. One way to do this is to use the same software.

The first motivation requires only an agreement on the communal methodology and terminology, so that we can all describe our experiments in comparable terms. The second motivation requires a body of portable code which can be made accessible to the general community. This body of code should be modularized in such a way that interested users can use those components which they find useful, and ignore others. For example, a researcher investigating a new domain might want just a utilities layer, and might wish to implement his/her own system and environment simulations using that. An implemented *S-E* language on top of the utilities

layer might enable such a researcher to define and construct new problem domains and IRTPS systems more rapidly, and might be attractive to a researcher if the modeling language seemed sufficiently expressive and concise. Finally, the third motivation above is best served by a shared implementation, at least up to the problem layer.

5.3 Experimental Methodology

We refer the reader to the included report from the working group on experimental methodology. Only a few additional comments will be made here. First, we noted earlier that there is a potential difficulty in obtaining significant results from experiments. Experiments in a simulated domain are of research interest only if:

1. the domain is intrinsically of interest to some community, or
2. strong arguments can be made that the domain is isomorphic to or a generalization of a large class of real world problems, or
3. we admit that we know so little that any bit of exploration is worth doing.

Domains of the first type are likely to be of interest only to a small segment of the potential IRTPS research community. On the other hand, it may be difficult to establish that a domain is of type two, given our current lack of knowledge about IRTPS in general. A testbed, however, might be helpful in validating such an hypothesis.

The consensus of the working group was that the ideal was a problem domain with as wide a "coverage" of IRTPS issues as possible. Further, the domain simulation should permit the researcher to adjust the "knobs" on the simulation to vary the complexity or stress along a variety of axes to meet current experimental needs. However, most felt that it was premature to claim that a single domain, however abstracted and generalized, could offer sufficient breadth of coverage, achieve widespread acceptance at this stage of IRTPS research, and avoid the danger of overlooking important issues.

There was general agreement that it would be a mistake to attempt to impose a single testbed or domain on the field. Rather, sharable code should be made available, and researchers should be permitted to chose from the available components, or construct their own experimental setups, as they choose. We did feel, however, that it would be appropriate to encourage at least some proposals in the next phase of the IRTPS initiative which included one or more of the following elements:

- Making existing software available for distribution.
- Adopting existing testbed software from another site, and incorporating that software into an IRTPS research program.
- Performing comparison studies within a single testbed.

We felt that these experiences would permit us to reconsider appropriate testbed methodology in the phase four workshop.

5.4 Analysis of Two Candidates

In this section we briefly review aspects of the Phoenix ([Cohen, Howe & Hart 90] (Appendix IX here)) and MICE ([Durfee & Montgomery 90] (Appendix X here)) testbeds. We chose these two because they were presented in readings to all the participants and because our group

included representatives from the groups developing them (Durfee for MICE and Howe for Phoenix). We made, and make, no attempt to evaluate either of these systems. The discussion below is organized around the testbed components identified earlier: platforms, utilities, *S-E* language, problem, knowledge, and behavior-metric mapping.

Platform

We discussed platform choices generally. There seemed near universal agreement that Common Lisp is the preferred implementation base. This bias certainly reflects the experience and familiarity of the participants. In particular, the unspoken assumption was that we are dealing with protoytype systems and there is a great need to build and evolve quickly, with little or no concern for actual fielding of systems.

CLOS (Common Lisp Object System) got a clear vote as the preferred object system for new implementations (although few people have used it as yet) and X seeming to be preferred as the host windowing system -- both of these because they are becoming standards and hence would support greater portability. No clear choice emerged for the Common Lisp interface to the windowing system, probably because there is no standard is yet emerging.

It also became clear that the lack of standardized multitasking facilities within Common Lisp would continue to be a serious impediment to sharing code. This is particularly problematic since the core of most testbeds seem to be some kind of discrete event simulator. The easiest and most efficient way to implement this is using multitasking, but multitasking primitives vary across Common Lisp and operating system implementations. However, this makes porting to other platforms difficult, and it does not readily allow for experimenting with systems whose target hardware architecture is different than that supported in the testbed, since the compute-time mapping may not be linear or even functional.

There seems to be a mix of Symbolics/TI and Unix hardware in the community. Most seemed to feel that porting code between Symbolics and TI would not present major obstacles in most cases, but between Symbolics/TI and Unix would be much more difficult. Phoenix was developed on the TI Explorer. MICE development was begun on the Apollo and later moved to TI Explorer.

Phoenix and DICE are both implemented in Common Lisp, both make minimal use of Flavors, and each uses its machine's native windowing environment. Phoenix uses the TI multitasking facility and makes extensive use of hooks into the scheduler to monitor and control resource allocation and simulate the temporal aspects. (See [Lark *et al* 90] for another example of such a simulation facility.) MICE does not use the TI multitasking facilities, but rather calls individual agent programs as functions, and expects them to report the amount of time spent problem-solving. This makes the MICE testbed itself simpler and quite portable, and also allows for the experimenter to impose whatever accounting policies he or she wants; but it leaves to the experimenter the chore of implementing those accounting and synchronization facilities and adding the various reporting information within all the system agents, most likely with a higher performance overhead.

Utilities

Both Phoenix and MICE are based on discrete event simulators (DESs). As described above, the Phoenix simulator relies heavily on TI Explorer multitasking and directly monitors cpu time used by a system as its measure of "cognitive time" of the system. MICE implements multitasking explicitly, and relies on the system to report time spent. The Phoenix DES is quite separable from the rest of Phoenix.

Both Phoenix and MICE provide some experimenter and system interface utilities. It is not clear to what extent these facilities are generic and extractable from the domains for which Phoenix and MICE are intended. Phoenix provides some automated logging facilities as well. Again, it is not clear how useful these are outside the firefighting domain.

Language

With regard to the *language* for describing the system, both Phoenix and MICE provide only a thin layer, not clearly distinct from the problem (see next). Each provides primitives for a spatial environment and expects the system to function in it. For example, each requires a notion of the location of each agent.

To configure an experiment in Phoenix, the developer defines which agents are present and selects their capabilities. The definition of capabilities (sensors, effectors, reflexes, and the cognitive components) is expressed at a lower level. Defining these is currently an art form, and the language used for this, which includes methods, mixins, Lisp code, and other declarations, is constantly evolving. This language is currently implemented within their custom-built frame language, but could be reimplemented easily in CLOS or other suitable tools, if desired.

For MICE, the developer defines the behavior of agents in "free form," providing operations to implement concepts such as location, orientation, sensors, movement, "linking," and colliding.

Both MICE and Phoenix "embed" the system in the environment, rather than modeling a totally symmetric interaction in which the environment would also be implemented as one or more agents. The feasibility and desirability of a symmetric representation and implementation are open questions. In both testbeds, then, the system interface provides access to both the utilities and the environment.

Problem

Phoenix presumes the firefighting problem. Specific scenarios within this problem domain are available and others can be defined by the experimenter.

MICE makes no explicit commitment to a specific problem. But it does assume that two dimensionality is an important aspect of the environment, which mediates interactions between the system and the environment.

The Phoenix firefighting problem and the generic MICE pursuit problems are similar in that they deal with two-dimensional simulated worlds and they treat time rather discretely. One participant has looked at applying a discrete event sumulator working in a mapped environment

to process monitoring tasks, and found difficulties. In particular, he found the need to replace the map by some data-oriented structures.

Both of these problems are also relatively simple, as contrasted, say, with the Pilot's Associate problem [Smith & Broadwell 88]. A worthy goal would be to have an *e. coli* problem -- one which has enough complexity to test the important ideas, but simple enough to permit relatively easy experimentation. We do not know enough yet to characterize such a problem for IRTPS, and indeed both the Phoenix and MICE efforts are attempting to evolve their problem domains in response to better understanding of those characteristics. See Section 3 for more discussion of related issues.

Knowledge

Much of the Phoenix agent knowledge was drawn from a text on firefighting policy. Phoenix represents some of this knowledge explicitly in the declarative plan libraries which the agents use.

MICE does not as yet have extensive development of any one specific problem domain, so discussion of available knowledge is inappropriate. Any such knowledge currently is implemented as straight Common Lisp code.

Behavior-Metric Mapping

Little has yet been done in either Phoenix or MICE to map from behavior to utility criteria. The Phoenix effort is just beginning to address this.

5.5 Summary

In summary, we came to the following conclusions:

- An experimental methodology is both desirable and possible for IRTPS.
- Establishing a common framework for discussing and sharing testbed components is important, for result sharing, reducing new research group startup costs, and enabling "distributed experimentation."
- An attempt to impose a single common testbed would be both inappropriate and doomed to failure. Rather, researchers should see the desirability of making use of common methodologies, tools, and problems. However, funders should directly support the creation of some testbed software, and should encourage its off-site reuse. This encouragement could take the form of evaluating more positively proposals which include offers to share testbed components as well as those which propose to adopt existing components.

Bibliography

[Agogino & Ramamurthi 88]
Agogino, A. M. & K. Ramamurthi. Real Time Influence Diagrams for Monitoring and Controlling Mechanical Systems, In *Proc. of Conf. on Influence Diagrams for Decision Analysis, Inference, and Prediction*. Berkeley, CA, 1988. To appear as: Chapter 9, pp. 228-252 in *Influence Diagrams, Belief Nets and Decision Analysis*, John Wiley & Sons. Provides the decision algorithm for a milling machine application along with a complexity analysis.

[Agogino & Ramamurthi 89]
Agogino, A. M. & K. Ramamurthi. Real Time Reasoning about Time Constraints and Model Precision in Complex Distributed Mechanical Systems. In *Symposium on AI and Limited Rationality (AAAI Spring Symposium Series)*, pages 1-5. Stanford University, March, 1989. Outlines constraint and goal classification approach to meta- reasoning about deadlines. Examples from past research are presented.

[Agogino, Guha & Russell 88]
Agogino, A.M., R. Guha & S. Russell. Sensor Fusion Using Influence Diagrams and Reasoning by Analogy: Application to Machine Monitoring and Control. *Artificial Intelligence in Engineering: Diagnostics and Learning*. Computational Mechanics Publications, Southhampton, 1988, pages 333-357. Reasoning by analogy is used to classify tool wear from acoustic signals.

[Agogino, Srinivas & Schneider 88]
Agogino, A. M., S.Srinivas & K. Schneider. Multiple Sensor Expert System for Diagnostic Reasoning, Monitoring and Control of Mechanical Systems. *Mechanical Systems and Signal Processing* 2(2):165-185, 1988. Real time influence diagram application to the supervisory control of a numerically controlled milling machine.

[Agre 88] Agre, P. *The Dynamic Structure of Everyday Life*. PhD thesis, AI Lab, MIT, 1988.

[Agre & Chapman 87]
Agre, P. & Chapman, D. Pengi: an Implementation of a Theory of Activity. In *Proc. National Conf. on Artificial Intelligence*, pages 268-272. 1987.

[Albus 85] Albus, J. *Brains, Behavior and Robotics*. Byte Books, Chichester, England, 1985. Chapter 5 describes multi-layer finite state machines.

[Allard & Kaemmerer 87]
Allard, J. & W. Kaemmerer. The Goal/Subgoal Knowledge Representation for Real-Time Process Monitoring. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 394-398. 1987.

[Altman & Buchanan 87]
Altman, R. & B. Buchanan. Partial Compilation of Strategic Knowledge. In *Proc. National Conf. on Artificial Intelligence*, pages 399ff. 1987.

[Anderson 88] Anderson, J. R. *The Architecture of Cognition*. Harvard University Press, 1988.

[Andersson 87] Andersson, R. *Real Time Expert System to Control a Robot Ping-pong Player*. PhD thesis, Dept of Computer Science, University of Pennsylvania, 1987.

[Barachini 88] Barachini, F. PAMELA : A Rule-Based AI Language for Process Control Applications. In *Proceedings of 1st Inter. Conf. on Industrial and Engineering Applications of Artificial Intelligence*, pages 860-867. Tennessee, 1988.

[Barachini & Theuretzbacher 88]
Barachini F. & N. Theuretzbacher. The Challenge of Real-Time Process Control for Production Systems. In *Proc. National Conf. on Artificial Intelligence*, pages 705-709. 1988.

[Barachini, Mistelberger & Bahr 89a]
Barachini F., H. Mistelberger & E. Bahr. The Art of Parallel Pattern Matching. In *IJCAI-89 Workshop on Parallel Algorithms for Machine Intelligence and Pattern Recognition*. Detroit, MI., August, 1989.

[Barachini, Mistelberger & Bahr 89b]
Barachini F., H. Mistelberger & E. Bahr. A New Method for Parallel Pattern Matching. In *Proc. of IFIP 11th World Computer Congress*, pages 337-342. San Fransisco, 1989.

[Barnett 84] Barnett, J. A. How Much is Control Knowledge Worth? *Journal of Artificial Intelligence* 22(1):77-89, January, 1984. A relatively simple, but powerful example of the value of applying metareasoning to raise the probability that an appropriate order of strategies will be applied to a problem instance.

[Baum *et al* 89] Kaiser, K. L. Baum, D. Blevins & V. Jagannathan. Adapting the Blackboard Model for Cockpit Information Management. *Blackboard Architectures and Applications*. Academic Press, 1989, pages 481-500, Chapter 21note="ThispaperdescribesthePilot/Vehicleinterfaceissuesincombataircraft.

[Baum, Jagannathan & Dodhiawala 89]
Baum, L. S., V. Jagannathan & R. T. Dodhiawala. The Erasmus System. *Blackboard Architectures and Applications*. Academic Press, 1989, pages 347-370. This chapter describes the Erasmus blackboard architecture and discusses the advantages of configurability.

[Boddy & Dean 89]
Boddy, M. & T. Dean. Solving Time-Dependent Planning Problems. In *Proc. Inter. Joint Conf. on Artificial Intelligence*. Detroit, MI, August, 1989. A discussion of limitations in classic AI planning techniques and of the application of flexible reasoning strategies, called "anytime algorithms," to the problem of planning under limitations in resources. Article also includes a description about a research direction focused on the application of decision theory to control reasoning.

[Bonissone 87a] Bonissone, P. P. Summarizing and Propagating Uncertain Information with Triangular Norms. *Inter. Journal of Approximate Reasoning* 1:77-101, 1987. Describes methods used in the RUM system, developed at GE's Corporate R&D center.

[Bonissone 87b] Bonissone, P. & K. Decker. RUM: A Layered Architecture for Reasoning with Uncertainty. In *Proc. Inter. Joint Conf. on Artificial Intelligence*. 1987.

[Bratman 87] Bratman, M. *Intention, Plans, and Practical Reason*. Harvard Univ. Press, 1987.

[Bratman, Israel & Pollack 88]
Bratman, M. E., D. J. Israel & M. E. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence* 4(4):349-355, 1988. This paper presents a high-level specification of the practical-reasoning component of an architecture for a resource-bounded agent. This architecture allows for means-end reasoning, for the weighing of competing alternatives, and for interactions between these two forms of reasoning -- and it does so in a way that addresses the problem of resource boundedness. A major role of the agent's plans in this architecture is to constrain the amount of further practical reasoning needed.

[Brooks 86] Brooks, R. A. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* RA-2(1):14-23, 1986.

[Brooks 87] Brooks, R. *Planning Is Just a Way of Avoiding Figuring Out What to Do Next*. Technical Report Working Paper 303, M.I.T. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1987. A discussion of the fundamental goals of planning. Discussion directs research beyond complex deliberative machinery, and towards more tractable reactive reasoning systems.

[Browne *et al* 90] Browne, J. C., E. A. Emerson, M. Gouda, D. Miranker, A. K. Mok & L. Rosier. Bounded-Time Fault-Tolerant Rule-Based Systems. In *Goddard Conf. on Space Applications of Artificial Intelligence*. 1990. (to appear).

[Browne, Chenk & Mok 90]
Browne, J. C., A. Cheng & A. K. Mok. Computer-Aided Design of Real-Time Rule-Based Decision Systems. In *IEEE Trans. on Software Engineering*. 1990. (to appear).

[Chavez & Cooper 88]
Chavez, R. M. & G. F. Cooper. A Fully-Polynomial Randomized Approximation Scheme for the Bayesian Inferencing Problem. *Knowledge Systems Laboratory Technical Report 88-72* , October, 1988. A discussion of work on the analysis of a simulation approach to performing the probabilistic inference problem. The method gives a worse-case analysis of inference as a function of a variety of parameters from the probability model.

[Chen 85] Chen, D. Shallow Planning and Recovery Planning Based on the Vertical Decomposition of the Flight Domain. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 1064-1066. 1985. Design for a hierarchical aircraft controller, with horizontal layers for route planning, approach planning, and flight.

[Cohen 89] Cohen, P. R. Discovering Functional Relationships that Model AI Programs. 1989. This paper presents a view of AI that supports prediction, generalization, and evaluation of hypotheses; and task analysis and design. This view characterizes AI systems in terms of their architectures, environments, and behaviors; and seeks general rules that justify and explain why agents should be designed one way rather than another, given the environments in which they operate. These rules call upon functional relationships to justify architectural decisions. In overview, we begin with five methodological problems that necessitate a change in the way we do AI research. One conclusion is that we build systems without clearly understanding why they work (or don't work). I sketch a solution to this problem in Section 3, in which I propose to analyze intelligent agents in terms of their architectures, behaviors, and environments. A pervasive theme of this document is that we can discover functional relationships between agents' architectures and their behaviors, which allow us to systematically design agents to behave as desired in particular environments. This is in sharp contrast to the current practice of designing AI systems "as best we can," without any analysis of why they work or don't work. Section 4 is a brief survey of how other behavioral sciences predict and explain behavior and introduces some familiar examples of functional relationships. Section 5 shows how functional relationships facilitate the design and analysis of AI systems. One example shows how we can apply the theory of signal detection to the task of designing a plan-selection mechanism.

[Cohen & Day 88]P. R. Cohen & D. S. Day. The Centrality of Autonomous Agents in Theories of Action Under Uncertainty. *Inter Journal for Approximate Reasoning* , 1988. In this paper we discuss a class of tasks in which to study planning under uncertainty. We analyze the interaction between autonomous agents and uncertain environments, and review the Artificial Intelligence literature on planning from this perspective. Case studies from our own research on medicine and controlling simulated forest fires are presented to illustrate issues in planning under uncertainty, and methods for studying these issues.

[Cohen & Howe 88]
P. R. Cohen & A. E. Howe. How Evaluation Guides {AI Research.} *AI Magazine* 9(4):35-43, 1988. Evaluation should be a mechanism of progress both within and across AI research projects. For the individual, evaluation can tell us how and why our methods and programs work, and so tell us how our research should proceed. For the community, evaluation expedites understanding of available methods and so their integration into further research. In this paper, we present a five stage model of AI research and describe guidelines for evaluation that are appropriate for each stage. These guidelines, in the form of evaluation criteria and techniques, suggest how to perform evaluation. We conclude with a set of recommendations that suggest how to encourage evaluation of AI research.

[Cohen & Howe 89]
P. R. Cohen & A. Howe. Toward AI Research Methodology: Three Case Studies in Evaluation. *IEEE Trans. on Systems, Man and Cybernetics* 19(3):634-646, May/June, 1989. We describe the roles of evaluation in empirical AI research -- in an idealized cyclic model -- and in the context of three case studies. The case studies illustrate pitfalls in evaluation and the contributions of evaluation at all stages of the research cycle. We contrast evaluation methods with those of the behavioral sciences and conclude that AI must define and refine its own methods. To this end we describe several experiment "schemas" and many specific evaluation criteria; and we offer recommendations that we hope will encourage the development and practice of evaluation methods in AI.

[Cohen *et al* 89] P. R. Cohen, M. L. Greenberg, D. M. Hart & A. E. Howe. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine* 10(3):34-48, Fall, 1989. For architecture/system, see I.9. This article describes the underlying methodology and illustrates the architecture and behavior of Phoenix.

[Cohen, DeLisio & Hart 89]
P. R. Cohen, J. L. DeLisio & D. M. Hart. A Declarative Representation of Control Knowledge. *IEEE Trans. on Systems, Man and Cybernetics* 19(3):546-557, May/June, 1989. We describe an explicit representation of control called *strategy frames*. The control of several well-known expert systems can be described in terms of strategy frames, although their control is actually encoded in an interpreter. One advantage of strategy frames is that complex control strategies emerge from their interaction, so complex interpreters are not necessary. We illustrate this idea in the context of a process control problem.

[Cohen, Howe & Hart 90]
Cohen, P. R., A. E. Howe & D. M. Hart. Intelligent Real-Time Problem Solving: Issues and Examples. In *Intelligent Real-Time Problem Solving Workshop Report.* Cimflex Teknowledge Corp., January, 1990.

[Cooper, Horvitz & Heckerman 88]
Cooper, G. F., E. J. Horvitz & D. E. Heckerman. *A Model for Temporal Probabilistic Reasoning.* Technical Report KSL-88-30, Stanford University, Knowledge Systems Laboratory, Stanford, CA, April, 1988. A discussion of the problem of temporal reasoning under uncertainty. First the general problem of uncertain reasoning through time is introduced. After, assumptions that reduce the complexity of the analysis are presented. The notion of general and stereotypical temporal functions are introduced. Following the exposition of a theory for combining uncertain beliefs over time, the paper presents work on an implementation of the techniques. System output is included in the discussion.

[D'Ambrosio 88] D'Ambrosio, B. Process, Structure, and Modularity in Reasoning with Uncertainty. In *Proceedings of Fourth Workshop on Uncertainty in Artificial Intelligence.* AAAI, Minneapolis, MN, July, 1988. A discussion of the importance of considering details of the structure and use of the probabilistic model in reasoning under uncertainty.

[Dean 89] Dean, T. On the Value of Goals. In Tennenberg, J., J. Weber & J. Allen (editors), *Proceedings from the Rochester Planning Workshop: From Formal Systems to Practical Systems*, pages 129-140. AAAI, Rochester, 1989. An interesting discussion of the application of decision theory to planning. Discussion examines the assignment of utility to goals in planning systems.

[Dean 90] Dean, T. Intelligent Real-Time Problem Solving Workshop. In *Intelligent Real-Time Problem Solving: Workshop Report.* Cimflex Teknowledge Corp., January, 1990.

[Dodhiawala 89] Dodhiawala, R. T. Foreword to section on Blackboard Systems in Real-Time Problem-Solving. *Blackboard Architectures and Applications.* Academic Press, Boston, 1989. This article discusses ways in which a blackboard architecture may be extended to respond to real-time problem-solving. It discusses essential aspects of related work.

[Dodhiawala *et al* 89]
Dodhiawala, R. T., N. S. Sridharan, P. Raulefs. & C. Pickering. Real-Time AI Systems: A Definition and an Architecture. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 256-261. Detroit, MI, 1989. This paper gives a definition of real-time in terms of four aspects: speed of processing, responsiveness to incoming events, timeliness in meeting deadlines, and graceful adaptation to adapt to changing workload. A knowledge processing architecture, RT-1, is presented. RT-1 has features that address the real-time aspects along with performance metrics and measures to evaluate the architecture. Related work is discussed.

[Dodhiawala, Sridharan & Pickering 89]
Dodhiawala, R. T., Sridharan, N. S., Pickering, C. A Real-Time Blackboard Architecture. *Blackboard Architectures and Applications.* Academic Press, Boston, 1989, Chapter 10. This is an extension of [Dodhiawala *et al* 89]. The added portions include a section on hardware-software studies which led the authors to derive several solution candidates. These call for a knowledge-processing approach to complex real-time systems.

[Doyle 88] Doyle, J. *Artificial Intelligence and Rational Self-Government.* Technical Report CS-88-124, Carnegie Mellon University, 1988. A discussion of a distributed approach to rationality based on the fair, democratic interaction among independent processes. Paper describes a language for reasoning about important predicates and functions in a distributed rational system.

[Drummond 85] Drummond, M. Refining and Extending the Procedural Net. In *Proc. Inter. Joint Conf. on Artificial Intelligence.* 1985. A definition of a plan structure and temporal projection mechanism that can describe iteration with non-deterministic termination. The representation is a generalization of previous work on partially ordered plans, drawing on work in Petri net theory.

[Drummond 86] Drummond, M. A Representation of Action and Belief for Automatic Planning Systems. *Reasoning About Actions and Plans.* Morgan Kauffman, 1986, pages 189-211. A representation and analysis mechanism is given for iterative and disjunctive plans. The definition of operator application that is given as part of the analysis mechanism is the first reported in the literature that uses the "consistency maintenance" approach (since popularized by Ginsberg and Smith). There are some relationships between temporal projection and truth maintenance system "groundedness" (in the sense of Doyle) that are mentioned in this paper; the details are worked out in one of the references.

[Drummond 89] M. Drummond. Situated Control Rules. In *Proceedings of Conf. on Principles of Knowledge Representation and Reasoning*. Toronto, 1989. Systems based on classical planning wisdom must produce a plan before they can take action. The execution component of such a system behaves like a CPU executing a progam: a CPU needs a program before it can do anything; similarly, the execution component of a system based on classical planning needs a plan in order to take action. This paper liberates the execution component of a system from dependence on a plan by interpreting the set of operators given to a system in two different ways: first, as a non-deterministic control program to run; and second, as a "causal theory" to push through temporal projection. The output from temporal projection informs the execution component. As a result, action can be taken when planning hasn't been done. Increased planning simply provides increased behavioral robustness.

[Durfee 88] Durfee, E. H. *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers, 1988. Describes an approach for coordinating multiple problem solvers, where the problem solvers must reason about the timing of their interactions. Includes details of the incremental planning approach and how blackboard-based problem solvers can use this approach to solve problems and share results in a timely fashion.

[Durfee & Lesser 87]
Durfee, E. H. & V. R. Lesser. Planning to Meet Deadlines in a Blackboard-based Problem Solver. In Stankovic, J. & K. Ramamritham (editor), *Tutorial on Hard Real-Time Systems*, pages 595-608. IEEE Computer Society Press, 1987. Overview of techniques for making predictions in a blackboard system that is performing repetitive problem solving, and for using these predictions to meet deadlines. Implementation and evaluation in the DVMT simulator for monitoring vehicle movements.

[Durfee & Lesser 88]
Durfee, E. H. & V. R. Lesser. Incremental Planning to Control a Time-Constrained, Blackboard-Based Problem Solver. *IEEE Trans. on Aerospace and Electronic Systems* 24(5):647-662, September, 1988. Describes incremental planning techniques for blackboard control and shows how they have been extended to incorporate rough predictions about the time needed to generate a solution. Describes how the planner uses these predictions to develop lower quality solutions within time constraints, where the user supplies preferences about how to trade quality for time.

[Durfee & Montgomery 90]
Durfee, D. H. & T. A. Montgomery. MICE: A Flexible Testbed for Intelligent Coordination Experiments. In *Intelligent Real-Time Problem Solving: Workshop Report*. Cimflex Teknowledge Corp., January, 1990.

[Fehling 88] Fehling, M. R. & J. S. Breese. *A Computational Model for the Decision-Theoretic Control of Problem Solving under Uncertainty*. Technical Report Rockwell Technical Report 837-88-5, Rockwell International Science Center, April, 1988. An example analysis of the decision-theoretic control of robot planning that considers base-level as well as meta-level considerations. The article considers issues of decision making under limitations in knowledge and computational time.

[Fehling, Altman & Wilber 89]
Fehling, M. R., A. M. Altman & B. M. Wilber. The Heuristic Control Virtual Machine: An Implementation of the Schemer Computational Model of Reflective, Real-Time Problem-Solving. *Blackboard Architectures and Applications*. Academic Press, Boston, 1989, Chapter 9. This paper presents real-time issues related to resource-bounded problem-solving, emphasizing the need for control reasoning to balance the constant tension between achieving goals and increasing belief in information. A computational model is discussed, along with the Schemer architecture. Several incarnations of Schemer exist -- Heuristic Control Virtual Machine and Schemer-II.

[Firby 87] Firby, R. J. An Investigation into Reactive Planning in Complex Domains. In *Proc. National Conf. on Artificial Intelligence*, pages 202ff. 1987.

[Firby 89] Firby, R. J. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. thesis RR-672, Dept of Computer Science, Yale University, 1989.

[Fox & Smith 84] Fox, M. S. & S. F. Smith. ISIS -- A Knowledge-Based System for Factory Scheduling. *Expert Systems* 1(2):25-49, 1984. Developed at CMU's Robotics Institute.

[Franklin & Gabrielian 89]
Franklin, M. K. & A. Gabrielian. A Transformational Method for Verifying Safety Properties in Real-Time Systems. In *IEEE 10th Real-Time Systems Symposium*. Santa Monica, CA, December, 1989. Provides a transformational approach for verifying that an Hierarichal State Machine (HMS) specification of a real-time system does not violate logical or temporal safety conditions. HMS machines, introduced in [Gabrielian & Franklin 88], provides a formal approach for specifying causal models of real-time systems.

[Gabrielian & Franklin 88]
Gabrielian, A. & M. K. Franklin. State-Based Specification of Complex Real-Time Systems, In *IEEE 9th Real-Time Systems Symposium*, pages 2-11. Hunstville, AL, December, 1988. Introduces the notion of Hierarchical Multi-State (HMS) abstract machine as a formal mechanism for defining real-time systems. HMS machines can be used for causal modeling and planning for dynamic environments, where deadlines have been imposed on objectives. Main benefits of HMS machines are (1) orders of magnitude reduction in the number of states, (2) formal verification methods and (3) reusability of models as requirements change.

[Gabrielian & Franklin 90]
Gabrielian, A. & M. K. Franklin. Multi-Level Specification and Verification of Real-Time Software. In *12th Inter. Conf on Software Engineering*. Nice, France, March, 1990. Presents a fundamentally new approach to multi-level specification of real-time systems using HMS machines [Gabrielian & Franklin 88] that offers a significant degree of reusability and modularity of specifications. As a demonstration of the application of HMS machines to scheduling with deadlines, this paper provides a method for automatically deriving schedules for plans to meet a complex set of logical and temporal constraints.

[Gabrielian & Stickney 87]
Gabrielian, A. & M. E. Stickney. Hierarchical Representation of Causal Knowledge. In *Proc. WESTEX-87 IEEE Expert Systems Conf.*, pages 82-89. June, 1987. Presents an early formalization of specification methods of [Gabrielian & Franklin 88, Franklin & Gabrielian 89, Gabrielian & Franklin 90] for causal modeling of systems with temporal constraints. Discusses preliminary approaches for application of such causal models to planning and situation understanding.

[Gallanti *et al* 85] Gallanti, M., G. Guida, L. Spampinato & A. Stefinini. Representing Procedural Knowledge in Expert Systems: An Application to Process Control. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 345ff. 1985.

[Georgeff 88] Georgeff, M. P. An Embedded Reasoning and Planning System. In *Advanced Proceedings from the Rochester Planning Workshop*, pages 79-101. University of Rochester Computer Science Dept., Rochester, NY, October, 1988. The paper describes the need for embedded systems that can reason and plan in real-time. The author presents Procedural Reasoning System (PRS) as a candidate architecture to meet this need. PRS consists of a database of current beliefs, outstanding goals, knowledge areas which embody plans, intention structure which points to active knowledge areas, and an inference mechanism which manipulates these structures. Meta-level knowledge areas support goal management as well as guaranteeing reactivity. These and other features of PRS are described in the context of a diagnostics application called Reaction Control System.

[Georgeff & Ingrand 89a]
Georgeff, M. P. and F. F. Ingrand. Monitoring and Control of Spacecraft Systems Using Procedural Reasoning. In *Proceedings of the Space Operations-Automation and Robotics Workshop*. Houston, Texas, 1989 This paper examines the application of SRI's Procedural Reasoning system (PRS) to the handling of malfunctions in the Reaction Control System (RCS) of NASA's space shuttle. Using various RCS malfunctions as examples -- such as sensor faults, leaking components, multiple alarms, and regulator and jet failures -- it is shown how PRS manages to combine both goal-directed reasoning and the ability to react rapidly to unanticipated changes in its environments.

[Georgeff & Ingrand 89b]
Georgeff, M. P. and F. F. Ingrand. Decision-Making in an Embedded Reasoning System. In *Proc Inter Joint Conf. on Artificial Intelligence*. Detroit, Michigan, 1989. This paper describes some of the features of the Procedural Reasoning System (PRS) that enable it to operate efficiently in continuously changing environments. PRS is able both to perform goal-directed reasoning and to react rapidly to unanticipated changes in its environment. It includes meta-level reasoning capabilities, which can be tailored by the user, employing the same language used to describe domain-level reasoning. It has been applied to various tasks, including malfunction handling on the NASA space shuttle, threat assessment, and the control of an autonomous robot.

[Georgeff & Lansky 87]
Georgeff, M. P. & A. L. Lansky. Reactive Reasoning and Planning. In *Proc. National Conf. on Artificial Intelligence*, pages 677-682. Seattle, WA, 1987.

[Ginsberg 88] Ginsberg, M. Multivalued Logics: A Uniform Approach to Inference in AI. *Computational Intelligence* 4:265-316, 1988.

[Ginsberg 89] Ginsberg, M. The Computational Value of Nonmonotonic Reasoning. December, 1989.Unpublished Stanford memo, available from the author.

[Ginsberg 90a] Ginsberg, M. Bilattices and Modal Operators. In *3rd Conf. on Theoretical Aspects of Reasoning about Knowledge*. Asilomar, CA, Spring (to appear), 1990. This paper presents a formalization of modal operators that appears to admit an anytime implementation. The implementation is not continuous or monotonic in the sense discussed at the IRTPS workshop, but does converge to the correct answer in the large runtime limit. The paper depends on previous work in [Ginsberg 88]. For architecture/system, see I.7..

[Ginsberg 90b] Ginsberg, M. Negative Subgoals with Free Variables. *Journal of Logic Programming* , (in preparation -- preprints available from the author), 1990. This and [Ginsberg 89] suggest that the real use of nonmonotonic reasoning is to focus the inference process in a way that allows an agent to "jump" to a conclusion and modify it later if need be. The other paper paper is informal; this one is formal.

[Giralt, Chatila & Vaisset 84]
Giralt, G., R. Chatila, & M. Vaisset. An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots. In *Robotics Research, the First Inter. Symposium*, pages 191-214. MIT Press, 1984.

[Griesmer *et al* 84]
Griesmer, J. *et al.* YES/MVS: A Continuous Real Time Expert System. In *Proc. National Conf. on Artificial Intelligence*, pages 130ff. 1984.

[Hall *et al* 89] Hall, D., A. M. Agogino, W. Greiman, D. Olson, R. Paasch, A. Padagonkar & E. Schroeder. *A Fault Location System for a Time of Flight Detector Array*. Technical Report 2748 2, Lawrence Berkeley Lab., 1989. To appear in Computing in High Energy Physics, 1989. Describes diagnostic reasoning system for a 10,000 sensor array for the LBL fast particle physics lab. Important real-time problem solving issues are addressed. The application is particulary difficult because of the complexity and sensory input involved.

[Hanson & Mayer 88]
Hansson, D. & A. Mayer. The Optimality of Satisficing Solutions. In *Proceedings of Fourth Workshop on Uncertainty in Artificial Intelligence*. AAAI, Minneapolis, MN, July, 1988. An early discussion of the control of search with decision theory. The discussion highlights tradeoffs in the problem of search. The 8-puzzle is used as an example to elucidate the possibility of controlling search with decision theory. Empirical results of work on different dimensions of search are included.

[Harel *et al* 88] Harel, D. *et al.* STATEMATE: A Working Environment for the Development of Complex Reactive Systems. In *Proc. 10th IEEE Conf on Software Engineering*. 1988.

[Hartman & Tenenberg 87]
Hartman, L. & J. Tenenberg. Performance in Practical Problem Solving. In *Proc. Inter. Joint Conf. on Artificial Intelligence*. 1987. Defines a cost metric that allows the comparison of alternative problem solving strategies under different assumptions about the statistical properties of the problem domain.

[Hayes-Roth 85] Hayes-Roth, B. A Blackboard Architecture for Control. *Artificial Intelligence* 26(2):251-321, July, 1985. Describes the BB1 architecture.

[Hayes-Roth 89] Hayes-Roth, B. Intelligent Monitoring and Control. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 243-249. Detroit, MI, 1989. This paper describes how perception, cognition, and action must be integrated for reasoning about complex, real-time systems. The ideas are explored by extending the BB1 control architecture, and using the new architecture for a surgical intensive care patient monitoring system called Guardian. The primary extension to BB1 architecture is to introduce multiprocessing capability so as to perform sensing and perception/cognition tasks in parallel.

[Hayes-Roth 90a] Hayes-Roth B. Research on Intelligent Agents. In *Intelligent Real-Time Problem Solving: Workshop Report*. Cimflex Teknowledge Corp., January, 1990.

[Hayes-Roth 90b] Hayes-Roth, B. Architectural Foundations for Real-Time Performance in Intelligent Agents. *Real-Time Systems* 2(1/2), May, 1990. (in press). This includes a description of an architecture for IRTPS based on the BB1/BB* blackboard architecture.

[Heckerman & Horvitz 87]
Heckerman, D. E. & E. J. Horvitz. On the expressiveness of rule-based systems for reasoning under uncertainty. In *Proc. National Conf. on Artificial Intelligence*, pages 121-126. Seattle, Washington, July, 1987. An analysis of pragmatic issues surrounding the efficiency of the production-rule versus the belief-network representation approaches for the problem of performing diagnosis under the general situation of uncertainty. The production-rule representation is shown to be limited in its ability to express efficiently important independencies and patterns of independence among distinctions in a knowledge base.

[Heckerman, Breese & Horvitz 89]
Heckerman, D. E., J. S. Breese & E. J. Horvitz. The Compilation of Decision Models. In *Proceedings of Fifth Workshop on Uncertainty in Artificial Intelligence*. AAAI, Windsor, Canada, August, 1989. A discussion of the optimal compilation of a decision problem, given (1) the stakes of a decision problem, (2) the cost incurred with computational delay, (3) a characterization of possible observations by a distribution over evidence weights, and (4) the cost of memory. The foundations of decision-theoretic compilation are introduced. Design-time selection rules for configuring optimal sets of rules are described. Analyses for different prototypical distributions of evidence weights are presented. Finally, the paper discusses issues surrounding the relaxation of assumptions, and highlights the usefulness of developing mixed compiled-compute strategies.

[Heckerman, Horvitz & Nathwani 89]
Heckerman, D. E., E. J. Horvitz & B.N. Nathwani. Toward Normative Expert Systems: The Pathfinder Project. *Computers and Biomedical Research* , (in press), 1989. A detailed review of the Pathfinder project to construct large probabilistic knowledge bases that contain generalized dependencies, and to perform inference tractably within these large models. A highlight of this work centers on the use of probabilistic similarity networks, a theoretically sound representation for focusing the attention of knowledge representation.

[Heckerman, Ng & Nathwani 89]
Horvitz, E. J., D. E. Heckerman, K. C. Ng & B. N. Nathwani. Heuristic Abstraction in the Decision-Theoretic (Pathfinder) System. In *Proceedings of the 13th Symposium on Computer Applications in Medical Care*, pages 178-182. Baltimore, MD, October, 1989. Describes the graceful modulation of abstraction as a useful class of partial computation. Abstraction of decision model is used to gain clarity and greater tractability. A difficult decision problem is decomposed into a hierarchy of goals at successively greater detail. In particular, the paper reviews the complexity of complete value-of-information analyses in the Pathfinder reasoning system, and show how the the evidence-gathering formalism can be applied to abstractions. The use of multiple abstraction hierarchies to control the focus of attention of a decision-theoretic analysis is introduced. Alternative hierarchies allow a problem to be probed and solved from different perspectives. Details on the implementation and interface associated with abstraction modulation in the Pathfinder system are presented.

[Hendler & Sanborn 87]
Hendler, J. & J. Sanborn. A Model of Reaction for Planning in Dynamic Environments. In *Proc DARPA Knowledge-Based Planning Workshop, Austin, TX*, pages 24.1-24.10. 1987.

[Herman & Albus 87]
Herman, M. & J. Albus. Real-Time Hierarchical Planning for Multiple Mobile Robots. In *Proc DARPA Knowledge Based Planning Workshop, Austin TX*, pages 22-1ff. 1987.

[Hewett & Hayes-Roth 88]
Hewett, M. & B. Hayes-Roth. Real-Time I/O in Knowledge-Based Systems. In *Proceedings of the Second Blackboard Workshop*, pages 107-118. AAAI, St. Paul, MN, August, 1988. Also published in *Blackboard Architectures and Applications*, Chapter 12, Eds. V. Jagannathan, R. T. Dodhiawala, & L. Baum, Academic Press, 1989. The authors present alternative extensions for asynchronous communication in the BB1 architecture. Each potential extension is evaluated in terms of several parameters. The implementation, along with a performance evaluation of the selected approach is presented.

[Horvitz 86a] Horvitz, E. J. Toward a Science of Expert Systems. In T. J. Boardman (editor), *Proceedings of the 18th Symposium on the Interface of Computer Science and Statistics*, pages 45-52. American Statistical Association, Ft. Collins, Colorado, March, 1986. Also available as Technical Report KSL-86-75, Knowledge Systems Laboratory: Stanford University, March 1986. Examines the promise of reasoning about computational and engineering tradeoffs to optimize the value of reasoning systems. The paper highlights the value of approximations for representations and computation that exhibit graceful degradation of performance, with decreasing allocation of engineering or reasoning resources. Formal research on the graceful degradation of value-of-information computation for use in the Pathfinder expert system is presented. This approach is based on reformulating information theory to handle group of diseases, versus single diseases, under consideration in a medical expert system.

[Horvitz 86b] Horvitz, E.J. *Reasoning about Inference Tradeoffs in a World of Bounded Resources.* Technical Report KSL-86-55, Stanford University, Knowledge Systems Laboratory, Stanford, CA, September, 1986. An early description of the use of multiattribute decision theory to control partial-computation strategies. The extensions of decision theory to include metareasoning about base-level inference are highlighted. A set of bounded-resource desiderata are enumerated for stategies that are used to solve problems under situations of varying or uncertain resource constraints.

[Horvitz 87a] E.J. Horvitz. Reasoning about Beliefs and Actions under Computational Resource Constraints. In *Proceedings of Third Workshop on Uncertainty in Artificial Intelligence*. AAAI, Seattle, Washington, July, 1987. Also in L. Kanal, T. Levitt & J. Lemmer, (ed.) *Uncertainty in Artificial Intelligence 3*, Elsevier, pps. 301-324. Early discussion of problems with traditional conception of rationality from decision science, as well as with heuristic solutions in AI research. Both naive decision-theoretic approaches and heuristic, satisficing approaches are viewed as suboptimal and costly. The paper introduces the decision-theoretic control of problem solving as a general framework for attacking the problem of ideal bounded rationality. Several problems, such as ideal compilation and metareasoning tractability are discussed; relevant literature from the decision sciences is highlighted. Concept of "bounded optimality," the optimization of the expected value of a reasoner, under constraints in its reasoning resources and constitution, is presented. Finally, properties desired of partial-computation strategies for reasoning under bounded resources are outlined. Properties of flexible computation include continuity, monotonicity, and convergence. These properties of flexibility are implicit in strategies that later came to be called "anytime algorithms.".

[Horvitz 87b] Horvitz, E. J. PRoblem-Solving Design: Reasoning about Computational Value, Tradeoffs, and Resources. In *Proceedings of the NASA Artificial Intelligence Forum*, pages 26-43. National Aeronautics and Space Administration, November, 1987. Also available as Technical Report KSL-87-64, Knowledge Systems Laboratory, Stanford University, October 1987. Discussion of the decision-theoretic design of problem solving versus real-time decision-theoretic control. Control reasoning is divided up into "strategic" and "structural" control; control based on determining a best order to apply well-defined computational policies or "strategies," is distinguished from control centering on the manipulation of the microstructure of an algorithm, to optimize its performance in some context. Finally, the paper discusses the feasibility of developing a formal science of limited rationality, by developing tools for design and control of computation within the framework of decision theory. Problems in reasoning under complexity and constraints of medicine are described.

[Horvitz 88] Horvitz, E.J. Reasoning Under Varying and Uncertain Resource Constraints. In *Proc. National Conf. on Artificial Intelligence*, pages 111-116. Minneapolis, MN, August, 1988. Formalizatic partial computation under bounded resources. The paper introduces a multiattribute-utility approac ial results for the control of fundamental computer-science problems, such as searching and sorting, and describes computational trajectories through multidimensional approximation spaces. A mathematical exposition highlights the value of strategies that exhibit monotonic refinement with the computation, for reasoning under varying and uncertain resource constraints. Includes an expected-utility analysis of flexible reasoning strategies under different classes of resource constraints. Resource classes include the situations of urgency, deadline, uncertain deadline, and urgent-deadline (a combination of the urgency and deadline situations). Beyond theoretical work, the paper describes empirical results from the performance of the Protos/Algo system, on the decision-theoretic control of fundamental computation tasks.

[Horvitz 89a] Horvitz, E. J. Rational Metareasoning and Compilation for Optimizing Decisions Under Bounded Resources. In *Proceedings of Computational Intelligence 89*. Association for Computing Machinery, Milan, September, 1989. A discussion of the relationship between decision-theoretic control of reasoning and the compilation of reasoning. The paper reviews the challenging issues surrounding construction of integrated reasoning systems that allow for the compilation of portions of deliberative processes at the base-level and meta-levels. Assumes a basic metareasoning architecture and allows for the introduction of several classes of compiled knowledge. Finally, the paper presents a view of learning as an agent's attempt to optimize its reasoning in a specialized environment, through local and long-term compilation.

[Horvitz 89b] Horvitz, E. J. *Some Fundamental Problems and Opportunities from the Standpoint of Rational Agency*. Technical Report KSL-89-28, Stanford University, March, 1989. A review of some challenging problems and research opportunities in the pursuit of "bounded-optimal" agents under resource constraints, presented at the AAAI Spring Symposium on Limited Rationality at Stanford, March, 1989. Issues discussed include problems at the foundations of preference regarding the optimization of long-term versus myopic utility, difficulties with evaluating the relative performance of different agents, problems associated with analytic regress, difficulties with the ideal integration of compiled and deliberative machinery, and problems with the analysis and validation of optimal partial-computation strategies.

[Horvitz, Breese & Henrion 88]
Horvitz, E.J., J. S. Breese & M. Henrion. Decision Theory in Expert Systems and Artificial Intelligence. *Inter Journal of Approximate Reasoning* 2 Special Issue on Uncertain Reasoning:247-302, 1988. A detailed review article, designed for the AI researcher, on past efforts, current studies, and the future promise of decision-theoretic principles for solving difficult AI problems. This article first presents mathematical principles and philosophical foundations of probability and decision theory. After the discussion of foundations, the paper provides a detailed review of applications of decision theory in AI. This includes both a historical study of problems and successes incurred in the past, and current research areas. Finally the article examines problems and directions for future research. The application of decision theory for controlling problem solving under bounded resources and for the optimization of plan optimization is highlighted.

[Horvitz, Cooper & Heckerman 89]
Horvitz, E. J., G. F. Cooper & D. E. Heckerman. Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 1121-1127. Detroit, MI, August, 1989. Introduces a formal model of rationality, based on the control of complex base-level inference by relatively tractable decision-theoretic metareasoning. The paper addresses the question, "How long should an agent deliberate about a problem before acting in the world?" The authors show that the answer to this question depends on (1) the stakes of the situation at hand, (2) the costs of deliberation, and (3) metaknowledge about the expected value of continuing to reason. In the general case, there may be uncertainty about all of these factors. Theoretical principles of belief and action under bounded resources are presented, and results of experiments with a complex decision problem, under resource constraints, are described.

[Horvitz, Heckerman & Langlotz 86]
Horvitz, E. J., D. E. Heckerman & C. P. Langlotz. A Framework for Comparing Alternative Formalisms for Plausible Reasoning. In *Proc. National Conf. on Artificial Intelligence*, pages 210-214. Philadelphia, Pennsylvania, August, 1986. A description of the relevance of uncertain reasoning to real-world problem solving. Fundamental properties of measures of belief are reviewed and are used to analyze alternative formalisms and approximations to probabilistic reasoning for pragmatic application in situations of limited information and computational resources. The relationships between probability theory and (1) the Mycin certainty-factor model, (2) fuzzy-set theory, (3) Dempster-Shafer theory, and (4) Endorsements are discussed.

[Horvitz, Suermondt & Cooper 89]
Horvitz, E. J., H. J. Suermondt & G. F. Cooper. Bounded Conditioning: Flexible Inference For Decisions Under Scarce Resources. In *Proceedings of Fifth Workshop on Uncertainty in Artificial Intelligence*. AAAI, Windsor, Canada, August, 1989. A description of a new flexible (or "anytime") algorithm for probabilistic inference called "bounded conditioning." The general probabilistic-inference problem is NP-Hard. This approach provides upper and lower bounds on a probability of interest; the bounds are tightened incrementally with computation, converging on a point probability with sufficient computation. The method is based on the decomposition of a difficult problem into a set of subproblems, by conditioning the problem on the truth of a spectrum of plausible contexts. The subproblems are solved in an order that guarantees that the most important aspects of the problem will be addressed first. The paper contains an analysis of why a great portion of the complete exponential problem is solved when only a fraction of the complete problem has been analyzed. In addition to the theoretical analyses, empirical study of the performance of the algorithm are described.

[Howe, Hart & Cohen 90]
Howe, A. E., D. M. Hart & P. R. Cohen. Addressing Real-Time Constraints in the Design of Autonomous Agents. *Real-Time Systems* 2(1/2), May, 1990. (in press). The Phoenix project is an experiment in the design of autonomous agents for a complex environment. The project consists of a simulator of the environment, a basic agent architecture, and specific implementation of agents based on real-time techniques; the first two parts have been constructed, the third is on-going. The facets of Phoenix that facilitate real-time research are: a simulator parameterized for varying environmental conditions and instrumented to record behaviors, an agent architecture designed to support adaptable planning and scheduling, and methods for reasoning about real-time constraints.

[Jagannathan, Dodhiawala & Baum 88]
Jagannathan, V., R. T. Dodhiawala & L. S. Baum. Boeing Blackboard System: The Erasmus Version. *Inter. Journal of Intelligent Systems* , 1988. Describes the Erasmus blackboard architecture, a configurable blackboard architecture derived from the BB1 architecture.

[Kaelbling 87a] Kaelbling, L. P. An Architecture for Intelligent Reactive Systems. *Reasoning About Actions and Plans*. Morgan Kaufmann, 1987, pages 395-410. Any intelligent agent that operates in a moderately complex or unpredictable environment must be reactive -- that is, it must respond dynamically to changes in its environment. A robot that blindly follows a program or plan without verifying that its operations are having their intended effects is not reactive. For simple tasks in carefully engineered domains, non-reactive behavior is acceptable; for more intelligent agents in unconstrained domains, it is not. This paper presents the outline of an architecture for intelligent reactive systems. Much of the discussion relates to the problem of designing an autonomous mobile robot, but the ideas are independent of the particular system. The architecture is motivated by the desires for modularity, awareness, and robustness.

[Kaelbling 87b] Kaelbling, L. P. Rex: A Symbolic Language for the Design and Parallel Implementation of Embedded Systems. In *Proceedings of the AIAA Conf. of Computers in Aerospace*. Wakefield, Massachusetts, 1987. A language for implementing embedded systems should facilitate the development of real-time programs, have clear formal semantics, allow both numeric and symbolic programming, and be easy to implement on parallel hardware. Rex, a Lisp-based language that has been used to implement a variety of mobile-robot control programs, satisfies these criteria and is a viable alternative for the implementation of embedded systems. Because Rex uses sequential circuits as its model of computation, it is easily adapted to run on parallel hardware, and may also be used to design custom chips.

[Kaelbling 88] Kaelbling, L. P. Goals as Parallel Program Specifications. In *Proc. National Conf. on Artificial Intelligence*, pages 60-65. 1988.

[Kaelbling & Rosenschein 89]
Kaelbling, L. P. & S. J. Rosenschein. Action and Planning in Embedded Agents. *New Architectures for Autonomous Agents (tentative title)*. Elsevier Science Publishers, 1989. Embedded agents are computer systems that sense and act on their environments, monitoring complex dynamic conditions and affecting the environment in goal-directed ways. Systems of this kind are extremely difficult to design and build, and without clear conceptual models and powerful programming tools, the complexities of the real world can quickly become overwhelming. In certain special cases, designs can be based on well-understood mathematical paradigms such as classical control theory. More typically, however, tractable models of this type are not available and alternative approaches must be used. One such alternative is the situated-automata framework, which models the relationship between embedded control systems and the external world in qualitative terms and provides a family of programming abstractions to aid the designer. This paper briefly reviews the situated automata approach and then explores in greater detail one aspect of the approach, namely the design of the action-generating component of embedded agents.

[Kaelbling & Wilson 88]
Kaelbling, L. P. & N. J. Wilson. *Rex Programmer's Manual*. Technical Note 381R, Artificial Intelligence Center, SRI Inter.,, Menlo Park, CA., 1988.

[Kaemmerer & Allard 87]
Kaemmerer, W. & J. Allard. An Automated Reasoning Technique for Providing Moment-by-Moment Advice Concerning the Operation of a Process. In *Proc. National Conf. on Artificial Intelligence*, pages 809ff. 1987.

[Laffey *et al* 88a] Laffey, T., P. Cox, J. Schmidt, S. Kao & J. Read. Real-Time Knowledge-Based Systems. *AI Magazine* Spring:27ff, 1988. This paper presents an overview of the issues in real-time AI systems. A definition of real-time is presented: the ability of the system to guarantee a response after a fixed time has elapsed. The paper then presents a good summary of various real-time AI applications and architectures, ranging from process control, aerospace, communications, medical, and robotics. The paper concludes with a set of theoretical issues in real-time knowledge-based systems.

[Laffey *et al* 88b] Laffey, T., S. Weitzenkamp, J. Read, S. Kao & J. Schmidt. Intelligent Real-Time Monitoring. In *Proc. National Conf. on Artificial Intelligence*, pages 73ff. 1988.

[Laird *et al* 89a] Laird, J. E., Swedlow, K. R., Altmann, E. & Congdon, C. B. *SOAR User's Manual: Version 5.0* 1989. In preparation. Manual for the latest version of SOAR.

[Laird *et al* 89b] Laird, J. E., Yager, E. S., Tuck, C. M. & Hucka, M. Learning in Tele-Autonomous Systems Using SOAR. In *Proceedings of the NASA Conf. on Space Telerobotics*. Pasadena, CA, 1989. In press. Using SOAR to control a hand-eye system (and to learn in the situation).

[Laird, Newell & Rosenbloom 87]
Laird, J. E., Newell, A. & Rosenbloom, P. S. SOAR: An architecture for general intelligence. *Artificial Intelligence* 33:1-64, 1987. Basic article on the SOAR architecture, which combines knowledge, reactivity, general problem solving, and learning.

[Laird, Rosenbloom & Newell 86]
Laird, J. E., P. S. Rosenbloom & A. Newell. Chunking in SOAR: The Anatomy of a General Learning Mechanism. *Machine Learning* 1:11-46, 1986.

[Langley *et al* 89] Langley, P., Thompson, K., Iba, W., Gennari, J. H. & Allen, J. A. *An Integrated Cognitive Architecture for Autonomous Agents*. Technical Report 89-28, Department of Information & Computer Science, University of California, Irvine, September, 1989.

[Lark *et al* 90] Lark, J. S., L. D. Erman, S. Forrest, K. P. Gostelow, F. Hayes-Roth, & D. M. Smith. Concepts, Methods, and Languages for Building Timely Intelligent Systems. *Real-Time Systems* 2(1/2), May, 1990. (in press). Describes the ABE/RT Toolkit -- a set of design, development, and experimentation tools for building time-stressed intelligent systems. The toolkit contains a set of languages for specifying the structure and behavior of timely systems, together with tools to simulate those models, log and analyze data collected during simulation runs, predict an application's performance on a specified target hardware architecture, and deploy the application on the target architecture. For architecture/system, see I.1..

[Lenat, Prakash & Shepherd 86]
Lenat, D. B., M. Prakash & M. Shepherd. CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. *AI Magazine* 6:65-85, 1986. Being developed at MCC.

[Lesser & Corkill 83]
Lesser, V. R. & D. D. Corkill. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine* 4(3):15-33, Fall, 1983.

[Lesser, Pavlin & Durfee 88]
Lesser, V. R., J. Pavlin & E. H. Durfee. Approximate Processing in Real-Time Problem Solving. *AI Magazine* 9(1):49-61, Spring, 1988. Also published in *Blackboard Architectures and Applications*, Chapter 11, Eds. V. Jagannathan, R. T. Dodhiawala, & L. Baum, Academic Press, 1989. The paper discusses three essential compromises a real-time problem-solver may make: certainty, precision, and completeness of the solutions. Approximations in solutions may be captured via data approximations (data abstraction) knowledge approximations (use partial knowledge) or approximate search strategies (reduce search space). The main thrust is to produce timely and satisficing solutions. The authors present various cases of these approximations in the Distributed Vehicle Monitoring System blackboard architecture, along with some experimental results.

[Long & Russ 83] Long, W. & T. Russ. A Control Structure for Time Dependent Reasoning. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 230ff. 1983.

[Marsh & Greenwood 86]
Marsh, J. & J. Greenwood. Real-Time AI: Software Architecture Issues. In *National Aerospace and Electronics Conf.*, pages 67-77. IEEE, Washington, D. C., 1986. This paper presents issues in real-time AI systems, but mainly from an 'implementation' perspective: memory management in Lisp, inefficiency due to interpretive languages for AI, poor software engineering, and so on. The authors define real time as 'predictably fast enough'. Issues in handling hard and soft deadlines are presented. But the main thrust of this paper is to push Ada over Lisp as a language for building real-time AI systems.

[Martin, Brown & Allen 88]
Martin, N., C. Brown & J. Allen. *ARMTRAK: A Domain for the Unified Study of Natural Language, Planning, and Robotics.* Technical Report, University of Rochster, 1988. ARMTRAK is a micro-world, based on the real-time control of model trains, designed to integrate work in natural language, planning, vision and robotics.

[Masui, McDermott & Sobel 83]
Masui, S., J. McDermott & A. Sobel. Decision Making in Time-Critical Situations. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 233-235. 1983.

[McCalla & Schneider 79]
McCalla, G. & P. Schneider. The Execution of Plans in an Independent Dynamic Microworld. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 553ff. 1979.

[McCalla & Schneider 82]
McCalla, G. & P. Schneider. Planning in a Dynamic Microworld. In *Proc. 4th CSCSI Conf*, pages 248ff. 1982.

[Mccalla *et al* 78] Mccalla, G., P. Schneider, R. Cohen, & H. Levesque. *Investigations into Planning and Executing in an Independent and Continuously Changing Microworld.* AI Memo 78-2, Department of Computer Science. University of Toronto, Toronto, Ontario, CANADA M5S 1A7, 1978. A taxi driver navigating through a simulated city with traffic & traffic lights -- simulated perception.

[McCalla, Reid & Schneider 82]
Mccalla, G., L. Reid & P. F. Schneider. Plan Creation, Plan Execution and Knowledge Acquisition in a Dynamic Microworld. *Int'l Journal of Man-Machine Studies* 16:89ff, 1982.

[Minton *et al* 87] Minton, S., J. G. Carbonell, D. Etzioni, C. A. Knoblock & D. R. Kuokka. Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System. In P. Langley (editor), *Proceedings of the Fourth Inter. Workshop on Machine Learning*, pages 122-133. Irvine, CA, 1987.

[Minton *et al* 89] Minton, S., J. G. Carbonell, C. A. Knoblock & D. R. Kuokka. The PRODIGY System: An Integrated Architecture for Planning and Analytical Learning. In K. VanLehn (editor), *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1989. In preparation.

[Miranker & Brant 90]
Miranker, D. P. & D. A. Brant. An Algorithmic Basis for Integrating Production Systems and Database Systems. In *Proceedings Sixth Inter. Conf. on Data Engineering*. February, 1990. Describes a lazy incremental match algorithm that folds the selection criteria of a forward-chaining production system into the search for satisfied rules such that only the winning instantiation is computed without exhaustively enumerating all the satisfied rules. Part way to an anytime algorithm for the incremental match problem.

[Mitchell *et al* 89] Mitchell, T. M., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette & J. C. Schlimmer. Theo: A Framework for Self-Improving Systems. In K. VanLehn (editor), *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1989. In press.

[Mitchell, Payton & Keirsey 87]
Mitchell, J., D. Payton, & D. Keirsey. Planning and Reasoning for Autonomous Vehicle Control. *Inter. Journal of Intelligent Systems* II:129ff, 1987.

[Newell, Rosenbloom & Laird 89]
Newell, A., Rosenbloom, P. S. & Laird, J. E. Symbolic architectures for cognition. In M. I. Posner (editor), *Foundations of Cognitive Science*, chapter 3. Bradford Books/MIT Press, Cambridge, MA, 1989. In press.

[O'Reilly & Cromarty 85]
O'Reilly, C. A., & A. S. Cromarty. 'Fast' is not 'Real-Time' in Designing Effective Real-Time AI Systems. In *Applications of Artificial Intelligence II*, pages 249-257. Int. Soc. of Optical Engineering, Bellingham, Washington, 1985. The authors motivate the need for real-time AI to handle the complexities and sophistication of the emerging applications in defense and industry. Real-time is defined as 'perceptually fast enough', to be comparable to human information processing capability. A formal definition of real-time is also given. Several optimization, scheduling, and search techniques are evaluated in terms of their real-time performance. Parallelism and metaplanning (or control reasoning) are suggested as solutions to build real-time AI systems.

[Pardee & Hayes-Roth 87]
Pardee, W. J. & B. Hayes-Roth. *Intelligent Real-Time Control of Material Processing*. Technical Report Technical Report #1, Rockwell International Science Center, 1987. This paper describes a material processing application using the BB1 architecture.

[Payton 86] Payton, D. An Architecture for Reflexive Autonomous Vehicle Control. In *IEEE Inter. Conf on Robotics and Automation*, pages 1838ff. 1986. Design for the ALV software, including a discussion of the assimilation/immediacy conflict as motivation for horizontal software layering.

[Payton 87] Payton, D. Autonomous Cross-Country Navigation with the ALV. In *Proc DARPA Knowledge Based Planning Workshop, Austin, TX*, pages 20-1ff. 1987.

[Pollack 89] Pollack, M. E. Plan Recognition Beyond STRIPS. In *IJCAI Workshop on Plan Recognition*. Detroit, MI, 1989. This short paper argues for integrating theories of real-time problem solving into models of plan recognition. In particular, it shows how the architecture for resource-bounded practical reasoning developed by Bratman, Pollack, and Israel (Computational Intelligence, 4:4), can lead to better systems for plan recognition.

[Powell & Cohen 89]
Powell, G. M. & P. R. Cohen. Operational Planning, Plan Monitoring and the PHOENIX System. In *under review for IEEE Annual AI Systems in Government Conf.*. George Washington University, Washington, D.C., May, 1989. Until recently computational models of planning have emphasized the generation of plans. Little attention was given to the processes of plan execution, monitoring and replanning. Clearly, when planning in unpredictable and dynamic environments, there is a significant requirement for effective plan monitoring and replanning capabilities. Operational planning is an exemplar of this class of problems. However, to our knowledge very little of the doctrinal or training literature addresses the problems and processes of plan monitoring and replanning. Consequently, the development of a model of operational planning presents a considerable challenge to our current computational understanding of planning. This paper presents a model of an AI planner for real-time control of fires forest, called PHOENIX, which generates plans, monitors them during execution, and revises them during execution when they go amiss. The paper focuses primarily on data structures called envelopes, which are critical to the activities of monitoring plans and projecting their progress. The utility of envelopes is illustrated in several examples from the PHOENIX environment and a hypothetical operational planning problem. We consider in detail the claim that the PHOENIX architecture is a good first attempt at an architecture for operational planning.

[Ramamritham & Stankovic 84]
Ramamritham, K. & J. A. Stankovic. Dynamic Task Scheduling in Distributed Hard Real-Time Systems. *IEEE Software* 1(3):65-75, 1984. An algorithm for scheduling tasks with hard deadlines is presented. It takes time to do the scheduling as well as resource constraints into account.

[Ramamurthi & Agogino 88]
Ramamurthi, K. & A. M. Agogino. Real Time Expert System for Fault Tolerant Supervisory Control. In *Proc. of ASME Inter. Computers in Engineering Conf.*, pages 333-340. 1988. Simulation study of a meta-controller for the supervisory control of a robotic manipulator. A real-time influence diagram architecture is used.

[Raulefs 89] Raulefs, P. Toward a Blackboard Architecture for Real-Time Interactions with Dynamic Systems. *Blackboard Architectures and Applications*. Academic Press, Boston, 1989, Chapter 13. This paper presents extensions to the blackboard architecture for improved reactivity. The author models each step of the typical top level control cycle as a set of concurrent processes, with emphasis on interruptability.

[Raulefs & Thorndyke 87]
Raulefs, P. & P. W. Thorndyke. An Architecture for Heuristic Control of Real-Time Processes. In *Proc. NASA/JPL Workshop on Space and Telerobotics*. NASA/JPL, January, 1987. This paper describe the HCVM architecture in the context of process planning and control application.

[Raulefs *et al* 87] Raulefs, P., B. D'Ambrosio, M. R. Fehling & S. Forrest. Real-Time Process Management for Materials Compositions in Chemical Manufacturing. In *IEEE Conf. on Applications of Artificial Intelligence*, pages 120-125. 1987. This paper describes the real-time issues in chemical process management. The HCVM blackboard architecture is described along with the Hueristic Process Control (HPC) layer on top of HCVM. The HPC framework captures aspects of the domain problem-solving. The Phosphorus Burden Advisor application is discussed.

[Rege & Agogino 86]
Rege, A. & A. M. Agogino. Sensor-Integrated Expert System for Manufacturing and Process Diagnostics. *Knowledge-Based Expert Systems for Manufacturing*. ASME PED, 1986, pages 67-83. A pump diagnostic application is developed in which real-time issues are addressed. Although not described in the paper, simulation software is available for testing IRTPS concepts like "time as a consumable resource" and "multiple failure modes".

[Rege & Agogino 88]
Rege, A. & A. M. Agogino. Topological Framework for Representing and Solving Probabilistic Inference Problems in Expert Systems. *IEEE Trans. on Systems, Man and Cybernetics* 18(3):402-414, 1988. Introduces the Berkeley IDES - Influence Diagram Based Expert System- architecture. Several IRTPS issues are addressed, including the ability to estimate accurately the computational time required to respond to any query through the "simul" module.".

[Retelle & Kaul 86]
LTC Retelle, J. P., Jr. & M. Kaul. The Pilot's Associate - Aerospace Application of Artificial intelligence. *Signal* :100-105, June, 1986. A technical description of the Pilot's Associate program is presented, along with reference to the approaches taken by the two prime contractors: Lockheed and McDonnell Douglas.

[Rosenbloom *et al* 89]
Rosenbloom, P. S., J. E. Laird, A. Newell & R. McCarl. A preliminary analysis of the foundations of the SOAR architecture as a basis for general intelligence. In D. Kirsh & C. Hewitt (editors), *Foundations of Artificial Intelligence*. MIT Press, Cambridge, MA, 1989. In press. Analysis of SOAR with respect to several aspects of general intelligence.

[Rosenschein 85] Rosenschein, S. J. Formal Theories of Knowledge in AI and Robotics. *New Generation Computing* 3(4, special issue on Knowledge Representation), 1985. Although the concept of *knowledge* plays a central role in artificial intelligence, the theoretical foundations of knowledge representation currently rest on a very limited conception of what it means for a machine to know a proposition. In the current view, the machine is regarded as knowing a fact if its state either explicitly encodes the fact as a sentence of an interpreted formal language or if such a sentence can be derived from other encoded sentences according to the rules of an appropriate logical system. We contrast this conception, the interpreted-symbolic-structure approach, with another, the situated-automata approach, which seeks to analyze knowledge in terms of relations between the state of a machine and the state of its environment over time using logic as a metalanguage in which the analysis is carried out.

[Rosenschein 89] Rosenschein, S. J Synthesizing Information-Tracking Automata from Environment Descriptions. In *Proceedings of Conf. on Principles of Knowledge Representation and Reasoning*. Toronto, Canada, 1989. This paper explores the synthesis of finite automata that dynamically track conditions in their environment. We propose an approach in which a description of the automaton is derived automatically from a high-level declarative specification of the automaton's environment and the conditions to be tracked. The output of the synthesis process is the description of a sequential circuit that at each clock cycle updates the automaton's internal state in constant time, preserving as an invariant the correspondence between the state of the machine and conditions in the environment. The proposed approach allows much of the expressive power of declarative programming to be retained while insuring the reactivity of the run-time system.

[Rosenschein & Kaelbling 86]
Rosenschein, S. J. & L. P. Kaelbling. The Synthesis of Digital Machines with Provable Epistemic Properties. In *Proceedings of the Conf. on the Theoretical Aspects of Reasoning About Knowledge*, pages 83-98. AAAI, Asilomar, California, 1986. Researchers using epistemic logic as a formal framework for studying knowledge properties of artificial-intelligence (AI) systems often interpret the knowledge formula $\mathbf{K}(x,\phi)$ to mean that machine x encodes ϕ in its state as a syntactic formula or can derive it inferentially. If $\mathbf{K}(x,\phi)$ is defined instead in terms of the correlation between the state of the machine and that of its environment, the formal properties of modal system S5 can be satisfied without having to store representations of formulas as data structures. In this paper, we apply the correlational definition of knowledge to machines with composite structure and describe the semantics of knowledge representations in terms of correlation-based denotation functions. In particular, we describe how epistemic properties of synchronous digital machines can be analyzed, starting at the level of gates and delays, by modeling the machine's components as agents in a multiagent system and reasoning about the flow of information among them. We also introduce Rex, a language for computing machine descriptions recursively, and explain how it can be used to construct machines with provable informational properties.

[Rosenschein *et al* 90]
Rosenschein, S. J., M. Fehling, M. Ginsberg, E. Horvitz & B. D'Ambrosio. IRTPS Workshop Interim Team Report. In *Intelligent Real-Time Problem Solving: Workshop Report*. Cimflex Teknowledge Corp., January, 1990.

[Rosenschein, Hayes-Roth & Erman 90]
Rosenschein, S. J., B. Hayes-Roth & L. D. Erman. Notes on Methodologies for Evaluating IRTPS Systems. In *Intelligent Real-Time Problem Solving: Workshop Report*. Cimflex Teknowledge Corp., January, 1990.

[Russell & Wefald 89]
Russell, S. J. & E. H. Wefald. Principles of Metareasoning. In R. J. Brachman, H. J. Levesque & R. Reiter (editor), *Proceedings of the First Inter. Conf. on Principles of Knowledge Representation and Reasoning*. Morgan Kaufman, Toronto, May, 1989. A review of earlier and ongoing work on the decision-theoretic control of problem solving in AI research. The article discusses several problems that pose challenges to the decision-theoretic control of game-playing search, the focus of the authors' research.

[Sanborn & Hendler 88]
Sanborn, J. & J. Hendler. A Model of Reaction for Planning in Dynamic Environments. *AI in Engineering* 3:2:95ff, 1988.

[Schoppers 87] Schoppers, M. Universal Plans for Reactive Robots in Unpredictable Environments. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 852-859. 1987.

[Schoppers 89a] Schoppers, M. *Representation and Automatic Synthesis of Reaction Plans*. PhD thesis, Dept of Computer Science, University of Illinois at Urbana-Chapmaign, 1989.

[Schoppers 89b] Schoppers, M. In Defense of Reaction Plans as Caches. *AI Magazine* Winter, 1989.

[Shoham 90] Shoham, Y. Agent Oriented Programming. In *Intelligent Real-Time Problem Solving: Workshop Report*. Cimflex Teknowledge Corp., Januar , 1990.

[Shoham & Hayes-Roth 90]
Shoham, Y. & B. Hayes-Roth. Report on Issues, Testbed, and Methodologies for the IRTPS Research Program. In *Intelligent Real-Time Problem Solving: Workshop Report*. Cimflex Teknowledge Corp., January, 1990.

[Skillman *et al* 89]
Skillman, T., W. Kohn, *et al*. A Class of Hierarchical Controllers and their Blackboard Implementations. *AIAA Journal of Guidance, Control and Dynamics* , May/June, 1989.

[Skillman, Kohn & Graham 88]
Skillman, T., W. Kohn, & R. Graham. Hierarchical Control of a Mobile Robot with a Blackboard Based System. In *Proc. Society of Photo-optical Instrumentation Engineers, Conf. on Mobile Robots III.* SPIE, Cambridge, MA, Nov, 1988. Design for a hierarchical/recursive robot controller, with horizontal layers from planning down to control.

[Smith & Broadwell 88]
Smith, D. & Broadwell, M. The Pilot's Associate -- An Overview. In *SAE Aerotech Conf.*. Los Angeles, CA, May, 1988.

[Sridharan & Dodhiawala 90]
Sridharan, N. S. & R. T. Dodhiawala. Real-Time Problem Solving: Preliminary Thoughts. In *Intelligent Real-Time Problem Solving: Workshop Report.* Cimflex Teknowledge Corp., January, 1990. Also appeared as FMC Corporation Corporate Technology Center, Report RT 2-89, July, 1989. The authors discuss pertinent issues in successfully building real-time AI systems, emphasizing the need for engineering approaches. The paper claims that none of the scientific methods of IRTPS will completely solve the IRTPS problem. Performance engineering real-time AI systems right from the problem formulation stage to the validation and testing stage is inevitable.

[Stankovic 88] Stankovic, J. A. A Serious Problem for Next-Generation Systems. *IEEE Computer* 10(21):10-19, 1988. The author argues that the next generation systems will be operating in severely time-stressed environments. The efforts of various research activities, including those in AI, databases, supercomputing, and operating systems, will be pooled to build the next generation real-time systems. The paper describes several misconceptions about real-time systems followed by issues in real-time architectures, databases, communications, and AI.

[Steier *et al* 87] Steier, D. M., Laird, J. E., Newell, A., Rosenbloom, P. S., Flynn, R., Golding, A., Polk, T. A., Shivers, O. G., Unruh, A. & Yost, G. R. Varieties of Learning in SOAR: 1987. In P. Langley (editor), *Proceedings of the Fourth Inter. Workshop on Machine Learning*, pages 300-311. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987. Overview of learning behaviors produced by SOAR.

[Swartout 87] Swartout, W. (ed.). Summary Report on DARPA Santa Cruz Workshop on Planning. In *Proc. DARPA Knowledge-Based Planning Workshop*, pages A.1-A.23. Austin, TX, 1987.

[Tambe *et al* 88] Tambe, M., D. Kalp, A. Gupta, C. L. Forgy, B. Milnes & A. Newell. SOAR/PSM-E: Investigating Match Parallelism in a Learning Production System. In *Proceedings of ACM/SIGPLAN symposium on Parallel Programming: Experience with Applications, Languages, and Systems*, pages 146-161. 1988. A parallel implementation of SOAR.

[Tambe, Newell & Rosenbloom 89]
Tambe, M., A. Newell & P. S. Rosenbloom. The Problem of Expensive Chunks and Its Solution by Restricting Expressiveness. 1989. Submitted to *Machine Learning*. An approach to bounding the computation performed during the match of productions.

[Tenenberg & Weber 90]
Tenenberg, J. & J. Weber. *A Statistical Approach to the Qualification Problem.* Technical Report, University of Rochster, (forthcoming), 1990. Describes the use of statistically-based probabilities for explicitly representing the uncertainty inherent in action reasoning.

[Unruh & Rosenbloom 89]
Unruh, A. & P. S. Rosenbloom. Abstraction in Problem Solving and Learning. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 681-687. Detroit, 1989. An approach to improving problem solving and learning through impasse-driven problem space abstraction.

[von Neumann & Morgenstern 47]
von Neumann, J. & O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ, 1947. The classic discussion of decision theory that presents that axioms of utility and describes a set of issues surrounding the application of probability and utility.

[Ward & McCalla 82]
Ward, B. & G. McCalla. Error Detection and Recovery in a Dynamic Planning Environment. In *Proc. National Conf. on Artificial Intelligence*, pages 172ff. 1982.

[Washington & Hayes-Roth 89]
Washington, R. & B. Hayes-Roth. Input Data Management in Real-Time AI Systems. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 250-255. Detroit, MI, 1989. This paper describes different approaches to reducing data glut to a real-time problem-solver. The limitations to conventional approaches are discussed. The authors then discuss dynamic thresholding as an approach. Data management options in the Guardian application are presented.

[Wefald & Russell 89]
E. Wefald and S. Russell. Estimating the Value of Computation: The Case of Real-Time Search. In *Working Notes: Spring Symposium Series on AI and Limited Rationality*, pages 106-110. AAAI, Stanford University, March, 1989. A discussion of techniques for estimating the value of computation for single-agent search.

[Weld 85] Weld, D. Combining Discrete and Continuous Process Models. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 140ff. 1985.

[Wellman 90] Wellman, M. P. *Formulation of Tradeoffs in Planning Under Uncertainty*. Pitman and Morgan Kaufmann, 1990. A detailed discussion of techniques for constructing simple decision models that can dictate optimal action under uncertainty, based on an analysis of fundamental tradeoffs.

[Wesson 77] Wesson, R. Planning in the World of the Air Traffic Controller. In *Proc. Inter. Joint Conf. on Artificial Intelligence*, pages 473-479. 1977.

[Whitehead & Ballard 89]
Whitehead, S. & D. Ballard. A Role for Anticipation in Reactive Systems that Learn. In *Proc. of Sixth Inter. Conf. on Machine Learning*. 1989. Integrates a forward (causal) model into a real-time, reinforcement learning control system in order to improve the system's ability to adapt its control policy.

[Williams 88] Williams, M. A. Hierarchical Multi-Expert Signal Understanding. In R. Engelmore & A. Morgan (editors), *Blackboard Systems*, pages 387-416. Addison Wesley, 1988. Discusses a signal understanding application, TRICERO. Various real-time issues are highlighted, including data glut, uncertainty, and timeliness of assessments.

[Wolfe 87] Wolfe, A. An Easier Way to Build a Real-Time Expert System. *Electronics Magazine* , March, 1987. Describes Gensym, Inc.'s G2 system.

I. Architectures and Systems

Presented here are brief descriptions of several architectures and implemented systems structures being used for IRTPS. This list, compiled from submissions by workshop attendees, while representative is certainly not comprehensive. The descriptions were, generally, supplied by individuals associated with the development of the respective architectures.

I.1 ABE/RT

The ABE/RT Toolkit is a set of design, development, and experimentation tools for building timely systems: time-stressed intelligent systems. The term *timely systems* is used to refer to systems with hard real-time requirements for interacting with a human operator or other agents with similar time-scales. The ABE/RT methodology is based on a philosophy of "rigorous engineering design" in which the application developer works to guarantee the system's timeliness by identifying the various events which require timely responses, determining the worst case frequencies of these events and the deadlines and durations of the tasks that respond to the events, and then verifying that the run-time system has enough processing resources to complete all mandatory tasks by their deadlines. The ABE/RT developers believe this is the only way in the near-term to build complex real-time intelligent systems reliable enough for critical applications with demanding users.

The ABE/RT Toolkit is being developed at Cimflex Teknowledge Corp. It contains a set of languages for specifying the structure and behavior of timely systems, together with tools to simulate those models, log and analyze data collected during simulation runs, predict an application's performance on a specified target hardware architecture, and deploy the application on the target architecture. [Lark *et al* 90] describes ABE/RT and its use for the Lockheed Pilot's Associate application.

I.2 DVMT

The Distributed Vehicle Monitoring Testbed has been developed at the University of Massachusetts since 1978. Its blackboard architecture has been extended with incremental planning control. This architecture extends traditional blackboard systems by incorporating a sophisticated planner to control the problem-solving activity. The planner uses approximate processing techniques to hypothesize possible solutions to pursue/refute, and plans knowledge source actions to efficiently perform this pursuit/refutation. This planning can incorporate time constraints by replacing time-consuming actions with less time-consuming (or no) actions, to achieve problem-solving deadlines at the price of reduced solution quality. The control mechanisms are built on top of the GBB (Generic Blackboard) shell, available from the University of Massachusetts. See [Lesser & Corkill 83, Durfee 88, Durfee & Lesser 87, Durfee & Lesser 88, Lesser, Pavlin & Durfee 88].

I.3 ERASMUS

Erasmus is a configurable blackboard architecture developed at Boeing's Advanced Technology Center, derived from the BB1 architecture [Hayes-Roth 85]. The Boeing group is using it for

our research in intelligent real-time systems. See [Jagannathan, Dodhiawala & Baum 88, Baum *et al* 89, Baum, Jagannathan & Dodhiawala 89].

I.4 The Situated-Automata Approach: GAPPS and REX

The major difficulty in programming intelligent real-time systems arises from the inherent tension between being "intelligent" (having the ability to recognize and respond appropriately to a wide range of situations) and being "real-time" (carrying out computations at the rate at which events are unfolding in the environment.) These two requirements are difficult to satisfy simultaneously, and current programming techniques usually achieve one at the expense of the other. Over the past several years, S. J. Rosenschein, L. P. Kaelbling, and other members of the situated-automata group at SRI International have been carrying out a program of research aimed at developing new techniques that combine the flexibility of symbolic reasoning systems with the performance of real-time control systems. By identifying and encapsulating certain abstractions relevant to the design of intelligent real-time systems, they hope to raise the conceptual level at which such systems are programmed and improve the efficiency of the programmer, as well as the capabilities of the target system. The situated-automata approach is not an "architecture" in the sense of a skeleton system with hooks for modules with defined semantic properties; it is a theoretical framework and an associated set of programming tools for facilitating the design process.

Theory: Situated automata research began by examining the foundations of AI knowledge representation systems to see whether symbolic reasoning, with its clear propositional semantics, could be theoretically reconciled with the demands of real-time performance. It was concluded that for applications in which systems interact with an external environment, it is not the symbolic expressions and their formal derivations, per se, that are of semantic significance, but rather the fact that these expressions are semantically interpretable by human designers and that they are objectively correlated with external reality. This viewpoint is elaborated in a mathematical framework called situated-automata theory [Rosenschein 85, Rosenschein & Kaelbling 86, Rosenschein 89]. In this framework, the concepts of mathematical logic are used to describe the external conditions with which machine states are correlated, but the theory does not require that the machine itself be modeled as an "inference engine." By liberating knowledge representation from the burden of run-time logical inference, the mathematical theory shows how machines can incorporate sophisticated semantic content and still operate in real time.

Programming Tools: A mathematical construction reconciling semantic complexity and real-time performance does not in itself enable intelligent real-time systems to be built in practice. Indeed, the practical problems in synthesizing systems for a particular task domain can often be quite daunting. Therefore a programming methodology and a set of software tools were developed that would ease the programmer's task while still producing provably real-time systems. As a concrete computational model, finite state machines were found to be convenient, particularly in their realization as sequential circuits; the principles of situated-automata theory give a compositional account of the semantic content of complex machine states in terms of their components, and the finite depth of the update and output circuitry place a hard bound on system response time.

Rather than asking the programmer to define this circuitry directly, however, symbolic

programming tools were developed that would indirectly define these functions in a way more readily interpretable by the programmer. The first of these tools was Rex (Kaelbling87b, KaelblingW88), a Lisp-based language in which the programmer could define symbolic programs that evaluated to circuits for extremely complex real-time perception and action systems. Although it was far easier to use Rex than it was to describe the circuit explicitly, Rex was still a low-level language in many ways, because the Rex programmer was more focused on the circuit itself than on the characteristics of the task domain. In an effort to raise the level of abstraction, a second tool was developed, Gapps (Kaelbling88), which allowed the programmer to write goal reduction rules and have them translated automatically to a provably real-time action-selection circuit.

I.5 HMS

Hierarchical Multi-State (HMS) machines, being developed at Thomson-CSF, are high-level automata that can be used for defining causal models of real-time systems. HMS machines can describe both real-time domains and problem solving strategies. Since temporal constraints can be defined and verified formally in HMS machines, they are particularly suitable for specifying IRTPS problems. HMS machines provide orders of magnitude reduction in number of states compared to traditional state-based representation methods and offer a promising approach to reusability and modularity of causal models. A number of applications of HMS machines, including hierarchical robot route planning, manufacturing scheduling and resource allocation in a dynamic environment with priorities have been considered so far. HMS machines are described in [Gabrielian & Stickney 87, Gabrielian & Franklin 88], FranklinG89, GabrielianF90]. Distributable software is not available at this time.

I.6 IRMA

The Intelligent Real-Time Machine Architecture (IRMA) is an architecture for the real-time practical-reasoning component of a resource-bounded agent [Bratman, Israel & Pollack 88]. It builds on a well-developed analysis of the role of an agent's plans in its further reasoning, first described in [Bratman 87]. More specifically, IRMA provides a framework in which an agent's plans help to constrain the amount of further practical reasoning the agent must perform, by focusing its subsequent means-end reasoning and by filtering out options that are incompatible with its existing plans. Implementation of agents embodying IRMA is currently in progress at SRI, being done by M. Pollack and M. Ringuette.

I.7 MVL

MVL implements the multi-valued logic work described in [Ginsberg 88, Ginsberg 90a]. The formalization of modal operators appears to admit an anytime implementation. The implementation is not continuous or monotonic in the sense discussed at the workshop, but it does converge to the correct answer in the large runtime limit. MVL is implemented in Common Lisp and is known to run on MacIntosh II and NeXT machines running Allegro Common Lisp, and Symbolics machines.

I.8 PAMELA

[Barachini & Theuretzbacher 88, Barachini 88] describe features of the PAMELA (PAttern Matching Expertsystem LAnguage) language. PAMELA includes a graphic development system for building "real-time" expert systems. The RETE-Algorithm is enhanced so that interrupts may be served at any time. Thus it is more sensitive to the environment than languages like OPS, ART, or G2. Also working memory elements may be modified outside the scope of rules. The graphic development environment is based on newest standards (X-Windows, DEC-Windows). The programming language is C, and in addition to a function-call interface to C, the user can program in "declarative C" style. PAMELA is currently available for research organizations and universities.

[Barachini, Mistelberger & Bahr 89a, Barachini, Mistelberger & Bahr 89b] describe research results on the parallel version of PAMELA for a distributed memory architecture.

I.9 Phoenix

Phoenix is a real-time, adaptive planner that manages forest fires in a simulated environment. Alternatively, Phoenix is a search for the functional relationships among the designs of agents, their behaviors, and the environments in which they work. In fact, both characterizations are appropriate and together exemplify a research methodology that emphasizes complex, dynamic environments and complete, autonomous agents. The Phoenix group, at the University of Massachusetts, is empirically exploring the constraints the environment places on the design of intelligent agents. See [Cohen *et al* 89, Howe, Hart & Cohen 90, Cohen & Day 88, Powell & Cohen 89] The Phoenix system comprises five levels of software: Discrete Event Simulator (DES), Map, Basic Agent Architecture, Agents and Agent Organization. The DES creates the illusion of a continuous world, where natural processes and agents are acting in parallel, on serial hardware. The Map level contains the data structures that represent the current state of the world as perceived by agents as well as the "world as it really is" and the methods that update the state of the world. The basic agent architecture provides the structure for sensors, effectors, reflexes and problem solving capabilities. The agent level describes the agents that we have designed for the Phoenix environment of forest fire fighting. The organization level includes a hierarchical organization of agents in which one fireboss directs multiple agents. The Phoenix environment (the DES and map level), the basic agent architecture and the agents with organization are independent software packages available for other researchers. The system runs in Common Lisp with Flavors on a TI Explorer.

I.10 PRS

The Procedural Reasoning System (PRS) is a generic architecture for real-time reasoning and acting that has been developed at SRI [Georgeff 88, Georgeff & Ingrand 89a, Georgeff & Ingrand 89b]. PRS is capable of operating efficiently in continuously changing environments. It can both perform goal-directed reasoning and react rapidly to unanticipated changes in its environment. It includes meta-level reasoning capabilities, which can be tailored by the user, employing the same language used to describe domain-level reasoning. PRS has been applied to various tasks, including malfunction handling on the NASA space shuttle, threat assessment, and the control of an autonomous robot.

I.11 RT-1

RT-1 is a small-scale, coarse-grained, distributed, event-driven architecture based on the blackboard paradigm. It consists of a collection of *reasoning modules* which share a common blackboard data space and operate asyncrhonously. Each reasoning module has 3 processes: the *I/O process* for inter- and intra-module communication, the *blackboard demon process* for asynchronous blackboard maintenance operations, and the *reasoning process* which performs the knowledge processing actions for the reasoning module. The main features of the reasoning process are: multiple, prioritized *event channels* for improved responsiveness to more critical events, explicit *meta-level reasoning* capability which permits opportunistic problem solving and ability to meet deadlines, and *execution margin* for intelligently regulating performance between the extremes of completely reactive to completely goal-directed according to changing workload and priorities. Thus, RT-1 addresses the responsiveness, timeliness, and graceful adaptation aspects of real-time as defined in [Dodhiawala *et al* 89]. RT-1 has associated with it measures and metrics that help evaluate the performance of the system.

I.12 SOAR

Soar is an AI architecture that combines a recognition-driven memory (a production system), the ability to interact directly with sensors and effectors, a decision cycle driven by the system's knowledge and perceptions, the ability to automatically generate subgoals, and the ability to learn from experience. Soar is written in CommonLisp, and runs on a variety of machines (Sun 3&4, Dec Vax & 3100, IBM RT, TI Explorer, ...), in CommonLisps provided by several different vendors. There is an extensive set of publications, including [Laird, Rosenbloom & Newell 86, Laird, Newell & Rosenbloom 87, Laird *et al* 89a, Rosenbloom *et al* 89, Steier *et al* 87, Laird *et al* 89b, Tambe *et al* 88].

II. Notes on Methodologies for Evaluating IRTPS Systems

Stanley J. Rosenschein (Teleos Research)
Barbara Hayes-Roth (Stanford University)
Lee D. Erman (Cimflex Teknowledge Corp.)

30 October 1989

Abstract

We propose a framework for modeling intelligent real-time problem-solving systems embedded in an environment. Within this framework, measurements may be defined on the system and on the environment, and particular measurements may be designated for judging the performance of the system. Although this framework supports analytical evaluation, we concentrate on its use for experimental evaluation, especially for evaluating and comparing system architectures. This framework also provides a basis for formalizing various requirements terms, such as "reactivity" and "graceful degradation".

1 Introduction

Intelligent real-time problem-solving systems (IRTPSs) are embedded computer systems that interact with their environments in a continuous fashion, sensing asynchronous events and acting in ways designed to satisfy certain goals. Instances of such systems include intelligent robots, factory control systems, avionic systems, and medical monitoring systems. Many software

*This work was supported by AFOSR Contract F49620-89-C-0129 and AFOSR Contract F49620-89-C-0117.

architectures have been proposed to ease the design and implementation of effective IRTPSs, and there has arisen the need for some objective means of evaluating and comparing them. As part of the research program on Intelligent Real-Time Problem Solving being sponsored by the Air Force, a small working group was formed to consider the methodologies for experimentally evaluating IRTPS architectures. This document is a preliminary report of that working group.

2 A Model of Embedded Systems

The first step in developing an evaluation methodology for IRTPSs is to lay out a conceptual framework for modeling systems and their interactions with the environment.

A natural beginning point is with models of dynamic physical systems. Such systems can be described in terms of time series of physical states, for example as mappings (possibly stochastic) from instants of time to some state space of values that model the physical features of interest. One then partitions the overall physical system into sub-components corresponding to the *IRTPS* S and the *environment* E, each having dynamic local state that varies as a function of signals received from the other. We may wish to regard the IRTPS and its environment as being parametrized in various ways. Let $S(u)$ and $E(v)$ represent the system and its environment with parameters u and v respectively. In addition, either or both of S and E could have random elements.

Because S and E are dynamic, time-varying objects, we are usually interested in describing not only those properties that hold statically of individual states, but properties of the time series of states. For present purposes, we will call these time series *runs* and write $runs(\langle S, E \rangle)$ to represent the set of runs of the combined IRTPS/environment pair. Each *run* is (conceptually) a sequence of total system states (i.e., states of the IRTPS/environment aggregate.) Conceptually, a run could be infinite and there could be an infinite number of runs.

Under this very general model, the boundary between S and E is arbitrary and can be adjusted to address different goals. In the present context, we are concerned with evaluating proposed IRTPS architectures with respect to their performance in particular classes of environments. Thus, the $S - E$

boundary should be placed so as to distinguish between a proposed architecture and the environment in which it is claimed to be effective. Then we can evaluate the relationships between properties of the architecture as manifested in S and properties of E. In particular, note that the $S-E$ boundary need not correspond to the boundary between a "complete agent" and its environment, but may correspond to the boundary around any "partial agent" of interest. For example, to evaluate a complete agent architecture, the $S-E$ boundary should encompass all perception, reasoning, and action elements. But to evaluate a perception architecture, the $S-E$ boundary should more tightly encompass only perceptual elements, with any reasoning or action elements treated as part of the environment. We might often choose to treat particular sensors and effectors as elements of E. As discussed below, for a given placement of the $S-E$ boundary, we will be attempting to attribute properties in the environment to the behavior of particular IRTPSs and, by inference, to their underlying architectures.

3 Measurement and Utility

In order to describe the effectiveness of an IRTPS architecture in controlling aspects of the environment, it is necessary to identify measurements that can be made and how those measurements will be interpreted to determine utility.

A measurement is any function of state values. Measurements can be made within a state or over sets of states or runs.

Under the above model of embedded systems, for a given $S-E$ boundary, measurements on E are distinguished from measurements on S. Measurements on E describe the dynamic properties of the environment, some of which are determined by processes internal to the environment and others of which are influenced by the behavior of S in E. The latter sorts of measurements are distinguished and used to assess the effectiveness of S in determining properties of E under various conditions. These assessments may be in absolute terms or relative to alternative Ss. Measurements on S describe the dynamic properties of the IRTPS, some of which are determined by processes internal to it and others of which are determined by the impact of E on S. These measurements are used help to explain the performance of S and its (in)effectiveness in determining properties of E in terms of its underlying ar-

chitecture. They also are used to analyze and predict its performance under other values of u and v.

Two classes of measurements are distinguished – descriptive and utility. Descriptive measurements represent objective features of a state, ru. or set of runs. Examples are: (a) deadline d was met; and (b) 80% of deadlines of type t were met. Other illustrative simple descriptive measurements are:

- Latency from environmental event e_1 to environmental event e_2 e.g., latency from occurrence of a fault to occurrence of its correction
- Deadline satisfaction e.g., whether a given fault is corrected by the time of its deadline
- Logical correctness of result
- Quality of result
- Precision of result

Illustrative functions on measurements are:

- Average latency for critical events
- Percent deadline satisfactions for critical events
- sum of latencies for all instances of event-type-a to event-type-b
- average percentage over deadline on soft-deadline events

Utility measurements represent valuational conclusions based on the features or qualities of a state, run, or set of runs. An example is: satisfactory performance requires meeting > 95% of priority 1 deadlines and > 50% of priority 2 deadlines. Other illustrative utility measurement (higher is better) are:

- Weighted sum of importance × deadline satisfied (0 or 1) for all events
- Weighted average of response quality × importance × deadline satisfied
- Gracefulness of degradation (suitably formalized)

The choice of utility measures may be specific to the $S - E$ boundary placement. They certainly will be specific to the purpose of the evaluation.

To describe system $S_1(u)$ parametrically with respect to various measurements of utility against fixed (parametric) environment $E(v)$ amounts to characterizing the expected utility of $S_1(u)$ as a function of (u, v), using a variety of techniques, some of which are described below. Similar methods can be used to compare two similarly parametrized systems, $S_1(u)$ and $S_2(u)$, fixed (parametric) environment $E(v)$.

4 Evaluation

In principle, there are many ways a proposed IRTPS design might be evaluated. The methods fall into two general categories: *analytical* and *experimental*. Because each of these methods has its advantages and drawbacks, a thorough evaluation often requires both. In the first section below, we briefly mention some of the advantages and disadvantages of analytical methods. The following section treats experimental methods, which are the focus of this document, in more detail.

4.1 Analytical Evaluation Methods

One method of characterizing system performance is by establishing certain of its properties through analytical techniques. These techniques draw on relevant mathematical methods and amount to proving theorems.

The main advantage of the analytical approach is the ability to establish with mathematical certainty very general or universal statements about entire classes of behaviors and phenomena far too numerous to enumerate in explicit detail. For instance, it might be shown mathematically that a certain undesired situation can *never* arise given the nature of the environment and the control system, or that when a triggering event occurs a response is *always* generated within a certain time period. Results of this kind can be very powerful and can give us great confidence in the systems we design.

Analytical approaches are not without their drawbacks, however. The primary drawback is that the complexity of the systems being modeled gives rise to intractable mathematical models that often resist analysis. In an attempt to render the models tractable, simplifications are often made which

cause the models to diverge from reality in ways that undercut the usefulness for building the model in the first place.

In defense of analytical methods it should be pointed out that there is no way to insure against bad modeling and analysis. Experience shows, however, that formal models are of use but often they must be supplemented by other techniques.

4.2 Experimental Evaluation Methods

The other main category of evaluation technique is experimental. In this approach, evaluation is done by measuring properties of particular instances of runs of particular systems and drawing certain conclusions. Experimental approaches can be further subclassified according to whether they involve (a) the real control system embedded in the real environment, (b) the real control system embedded in a simulated environment, (c) a simulated control system in a simulated environment, or (d) some hybrid approach. Real systems offer the obvious advantages of evaluating against reality, but they are often cumbersome or even unavailable, may pose unacceptable risks, etc. In the simulation approach to experimental evaluation, the environment and possibly the control system as well (e.g., if we are using a conventional machine to simulate a massively parallel system controlling an environment) a computer program is run to generate samples of the behavior of the overall system. These samples are then analyzed empirically to provide evidence in favor of certain conclusions regarding performance of the real system in the real environment.

Experiments provide a framework for inductive inference of general relations between architectures and real-time performance based on observations. The goal is to discover general relations that can be expected to hold whenever the appropriate conditions hold. For example, one relation might be that architecture A provides graceful degradation in performance under increasing rates of environmental events; another relation might be that architecture B produces unacceptable performance degradation under similar circumstances. Because it is infeasible to make observations of all of the instances encompassed by a hypothesized relation, it becomes necessary to draw conclusions about what would happen for all such instances based on observation of a few particular instances. For example, we might draw conclusions about the relative advantages of architectures A and B on the basis

of their performance on a small number of environmental scenarios.

Drawing general conclusions from a small number of observed instances is a risky business. A given S realizes an abstract architecture, A, in a particular implementation, I, and instantiates it for a body of knowledge, K. Architectures themselves are complex artifacts, differing in both theoretically interesting variables (e.g., knowledge representation, inference procedures, control mechanism) and incidental variables (e.g., implementation details, execution environment). Similarly, different environmental scenarios differ in a great many variables (e.g., frequency of important events, distribution of deadlines, amount of interpretation required, predictability of events). As a consequence, any given observation of the performance of a given architecture on a given environmental scenario is likely to reflect the combined effects of many such variables.

Controlled experimentation is an attempt to reduce as much as possible the incidental variability in a set of observations in order to: (a) obtain a reliable account of particular effects of a particular set of theoretically interesting variables; and (b) rigorously bound the class of situations in which those effects can be expected to obtain. In a controlled experiment, one or more "independent variables" are manipulated and their distinctive effects on one or more "dependent" variables are measured. In the present context, we will typically be manipulating independent variables representing a proposed architecture within S and measuring dependent variables representing the performance of interest within E. Other variables, including both S and E variables, are "controlled" to avoid confounding their effects with those of the independent variables. We also will often evaluate the performance of a fixed S as a function of various E scenarios; for this, we keep S fixed and the dependent and independent variables are in E.

Some controlled variables are simply held constant, while others are systematically manipulated or randomly sampled to provide a basis for generalization. In the domain of IRTPS systems, there are a variety of important variables we may wish to control:

S variables:

- sensors
- effectors
- knowledge available

- computing resources
- responsibilities

E variables:

- domain
- rate of critical/non-critical events
- distribution of deadlines
- complexity of events (e.g., multi-variate, temporal properties, noisy, uncertain)
- complexity of required effects of actions
- complexity of reasoning required
- knowledge required
- tasks required

In general, the more thoroughly variables are sampled within a class, the more reliably we can generalize conclusions based on associated observations. Statistical inference techniques permit probabilistic statements about the likelihood that relations observed in a given number and distribution of instances will hold for the entire class of such instances.

For example, to compare two different control mechanisms, A and B, we might set up two complete agent architectures differing only on this single independent variable, while holding constant all other architectural variables. In effect, we would be placing the $S-E$ boundary tightly around the control mechanism. We might then apply the architectures to a set of scenarios in which we hold constant the environmental domain (e.g., power plant monitoring) and systematically manipulate the frequencies of critical and non-critical events and the associated distributions of deadlines. In each condition (unique combination of values of independent and controlled variables), we would measure several dependent variables, such as logical correctness of response and satisfaction of deadlines for critical and non-critical events. Given an appropriate statistical analysis of the results of these measurements, we

might draw certain conclusions with high confidence, for example: (a) for critical events in all conditions, control mechanism A produces a higher rate of correct answers within deadline than control mechanism B (97% vs. 75%); (b) for non-critical events in all conditions, control mechanism B produces a higher rate of correct answers within deadline than control mechanism A (75% vs. 65%); (c) as the frequency of events increases (1→50 events per unit time), control mechanism A's performance on critical events degrades slowly (100→95%), while its performance on non-critical events declines dramatically 90→40%); and (d) as the frequency of events increases, control mechanism B's performance declines significantly for both critical and non-critical events (95→50% in both cases). Given a particular set of utility functions–in particular, valuing critical events more highly than non-critical events–we might conclude from these results that control mechanism A is "better" than control mechanism B because it provides "better" performance overall and a "better" degradation profile.

Control of variables determines what kinds of conclusions an experiment can support. For example, observing the performance of S_1 and S_2 on a single scenario, O_1, in a single environment, E_1, permits only conclusions about the comparative effectiveness of S_1 and S_2 on scenario O_1. Observing performance on a representative sample of scenarios in E_1 permits conclusions about the comparative effectiveness of S_1 and S_2 in environment E_1. Observing performance on a sample of scenarios in a sample of environments from class E permits conclusions about the comparative effectiveness of S_1 and S_2 in environment class E.

Of course some variables are quite difficult to control, and these necessarily limit the conclusions that can be drawn from experiments. In particular, it is difficult to separate and control variables that distinguish among the architecture, implementation, and knowledge of a given system S. Nonetheless, if we wish to draw strong conclusions about the utility of an architecture, we must control these potentially confounding variables. In many cases, our experiments may permit conclusions only about the level of performance of an S or the relative performances of alternative Ss. It may require analytical methods–or be infeasible–to determine whether an S's performance advantage is due to its architecture, implementation, or knowledge.

5 Other Implications

Although the primary purpose of this document concerns methodologies for evaluating IRTPS architectures, these considerations also carry implications regarding requirements specification and experimental testbed.

5.1 Implications for Requirements Specification

At this early stage of IRTPS research, we have many proposals for IRTPS requirements that use idiosyncratic, but overlapping vocabularies to describe concepts whose relationships to one another are ambiguous. The proposed model of embedded systems offers an opportunity to operationalize intuitive concepts like reactivity, coherence, interruptability, etc. in terms of basic measurements on state values. Thus, it will become clear, for example, that researcher A uses the term "interruptability" to refer to the latency to respond to a critical event, while researcher B uses the same term to refer to the ability to abort an ongoing computation. Although this does not insure agreement on terms and definitions, it provides a "lingua franca" for communication about terms and definitions.

The concept of an $S-E$ boundary is fundamental to the proposed model of embedded systems. Although placement of the boundary is flexible, to allow study of IRTPSs of differing scopes, a given placement of the boundary structures the definition of requirements and assessment of their satisfaction. Thus, on one side of the boundary, we have the S whose "requirements" constitute a theory or design that is hypothesized or intended to produce satisfactory consequences in E. On the other side of the boundary, we have the E whose "requirements" define the so-called satisfactory consequences. An informative experiment will tell us to what degree the requirements hypothesized for S (and realized in a particular implementation) actually achieve the requirements demanded for E.

For example, we call systems "reactive" in (at least) two different situations. First, we speak of reactive systems that iterate a highly efficient sense-act loop. Second, we speak of systems as reactive if they react promptly to important external events. Under the proposed model of embedded systems, the first sense of reactive is a hypothesized requirement on S, S-Reactivity while the second is a defined requirement on E, call it E-Reactivity. The testable claim is that S-Reactivity produces E-Reactivity (and perhaps some

other desirable E-Requirements as well). Notice, however, that other testable claims are possible, for example that a non-reactive S (one whose architecture is something different from the above-mentioned sense-act loop) also produces E-Reactivity (and perhaps some other desirable E-Requirements as well). The purpose of experiments is to evaluate such claims.

Accordingly, the model of embedded systems suggests that efforts to specify requirements for IRTPSs clearly distinguish between S-Requirements and E-Requirements. In particular, E-Requirements should be defined strictly in terms of measurements on E variables, while S-Requirements should be defined in terms of measurements on S variables. For example, S-Reactivity might be defined as a bound on the computation performed between sensing and acting. E-Reactivity might be defined as a bound on the latency between occurrence of an important "problem" event and the occurrence of an appropriate external "correction" event. Moreover, a given experiment requires agreement among participants on the E-Requirements against which alternative S-Requirements will be evaluated.

5.2 Implications for an Experimental Testbed

As discussed throughout this document, IRTPSs are complex artifacts embedded in complex environments. Experiments that allow generalization of conclusions beyond the immediate experimental conditions require control of many variables in both the S and the E. In addition, experimentation on S's of differing scopes requires flexibility in the placement of the $S-E$ boundary. To support these kinds of experimentation requires a sophisticated testbed.

For example, a basic testbed would allow one with an S to run and experiment with it. The testbed provides an E (likely simulated, and also perhaps paremeterized) and defines the $S-E$ boundary by the interface functions and data structures through which S and E interact. The testbed should also provide means for controlling multiple runs and for collecting measurements on E and, perhaps, on S as well. Ideally a testbed also provides utilities for analyzing the results (i.e., the measurements) across multiple runs.

A more general testbed facility would not have the E built in, but would instead accept both the E and the S as inputs. That is, it defines a generic interface between any E and any S compatible with that E. It would accomplish this by defining an interface between the testbed itself and S, and between the testbed and E. These interfaces would allow the testbed to

control each of S and E, to support their interactions, and to collect the measurements.

For still more flexibility, a testbed would support experiments with varying boundaries between S and E. That is, the experimenter would supply a complete, modular system to the testbed and specify, for any run, where the $S-E$ boundary is. This requires a more extended interface definition facility - one which supports a complex of interacting modules, preferably composable, and allows for measurements on any of their interfaces. Indeed, we can generalize this concept so that the $S-E$ boundary is not even fixed for a run, but shows up only in terms of which measurements are designated the utility measurements. This reflects the idea that the S and E together are a complete system, and specifying the boundary is only a means for analysis and evaluation.

III. Report on Issues, Testbed, and Methodology for the IRTPS Research Program

Report on Issues, Testbed, and Methodology for the IRTPS Research Program

Yoav Shoham
and
Barbara Hayes-Roth

Stanford University

October, 1989

1. Overview

AFOSR has posed three questions regarding intelligent real-time problem solving (IRTPS):

(a) What are the important terms and issues in IRTPS?

(b) What characteristics should appear in a community testbed application?

(c) What experimental methodology should guide research in this area?

It is easier to contrast an IRTPS system with other frameworks than to define it. IRTPS systems contrast on the one hand with traditional real-time systems, and on the other hand with classical AI planning systems. With the former they share properties such as responsiveness and timeliness. With the latter they share concepts such as process representation, prediction, and reasoning about resource limitations.

To actually define IRTPS systems we need to give precise meaning to the notions mentioned above, and to others such as reactivity, robustness, flexibility and uncertainty. In section 2 we propose a neutral setting in which these terms can be defined, which is compatible with the framework proposed in another workshop paper, "Notes on Evaluating Methodologies for IRTPS Systems," by Stan Rosenschein, Barbara Hayes-Roth, and Lee Erman. In section 3 we introduce several intuitive notions regarding IRTPS requirements and define them in terms of the proposed framework. In section 4 we briefly outline our particular perspective on IRTPS systems, that of *real-time intelligent agents* (on which we say more in two attached documents). In section 5 we give recommendations regarding the testbed environment. Our recommendations regarding experimental methodology appear in the above-mentioned paper and are not repeated here. In section 6 we offer a particular recommendation for structuring AFOSR's IRTPS Research Program.

2. A Computational Model for IRTPS Systems

This section discusses how the modeling of a computational device and its environment, and draws on an analogy with control theory.

Franz Reuleaux' book of 1876, The Kinematics of Machinery, is considered to have laid the foundations to modern kinematics. His insight was the following. In order to reason about the possible motions of physical objects in space, both rotation and translation, one should focus not on the single object but on *pairs* of adjacent objects, the *kinematic pair*. The fundamental object of investigation, proposed Reuleaux, was the interaction between physical objects, the *joint*. The basic question, given a joint between two objects in a specific geometric configuration, is what translational and rotational freedom they allow each other. Reuleaux was able to give a general answer to the question (which since then has been refined and extended).

A similar view of computation is possible, which focuses on the interaction of a machine with its environment. Furthermore, as in the case of kinematics, one need not speak of a machine as one sort of object and its environment as another; instead they can be viewed symmetrically as interacting machines, the *informatic-pair*. A theory of any sort of computation now becomes a theory about the interaction between machines.

A few simple definitions are probably in order at this point. We first define an automaton to consist of several inputs and outputs, an internal state, and a transition function. The transition function is the only nontrivial aspect of the definition: it specifies that the state and output of the machine at each time t are determined by its state at time t-*deltaS*, and its input at time t-*deltaI*. In the following we assume DT, a set of nonzero durations. In the context of an IRTPS system, we take DT to be the positive real numbers.

Definition 1. An *automaton* (also called a machine) is a tuple

$$[S, I, O, F, deltaS, deltaI]$$

where S is a set of states, I is a (possibly empty) set of n-ary tuples of input values (for some fixed n), O is a set of m-ary tuples of output values (for some fixed m), and F is a transition function $F: S \times I \rightarrow S \times O$, and *deltaS*, *deltaI* are respectively the state and input lags.

Thus, if the machine's state at time t-*deltaS* is s_1 and its input at time t-*deltaI* is i_1 and furthermore $F(s_1,i_1)=(s_2,o_2)$, then at time t the machine will be in state s_2 and have output o_2.

With the exception of the temporal aspect, this is a standard construct in system-modeling disciplines. It is a very general model. Although we

will restrict the discussion in this manuscript to finite and deterministic automata, we allow in general infinite sets of internal states and stochastic transition functions.

In general we cannot speak of the behavior of the automaton, since that depends on the input. The one case in which we can do it is when the set of inputs is empty.

Definition 2. An automaton with an empty set of inputs is called *closed*; otherwise it is *open*.

For a closed automaton we can assume a unary transition function, one that depends only on its internal state. Given an initial history of the closed automaton's internal states and outputs, we can define the *trace* [or *run* or *behavior*] of the automaton at all future times in the obvious way.

The systems we build, whether they are low-level process controllers or high-level planners, are open automata; we can model their behavior only in conjunction with their environment, which, as was mentioned, will be viewed as simply another automaton.

We first define what it means to *wire* together a collection of automata. Intuitively, when we wire such a collection we connect some inputs to some outputs: an input is connected only to an output of another machine, not necessarily all inputs and outputs are connected, and at most one output is connected to each input and vice versa. (It may seem natural to allow connecting more than one input to a single output. We disallow it because it makes for cleaner semantics of composite machines below. However, we allow a machine to have several outputs that always carry the same value.)

Such a wired set of automata induces a new automaton. Intuitively, the new states are the cross-product of the old states, the new inputs are the union of the old ones that have not been connected to an output, and the new outputs are the union of the old outputs that have not been connected to an input. The only nontrivial aspect is the definition of the new transition function. The subtlety is in the timing. Suppose we manage to catch the collection of automata at a moment when all of them just switched to a new state, which by definition is a new state for the composite machine. When is the next state-change of the composite machine? Intuitively, the composite machine will undergo state transitions at several times, corresponding to the different delays of the

individual machines. However, by definition any automaton has a single pair of delays associated with it. Indeed, we will define the composite automaton to change state as soon as any of the individual machines do. In other words, the delay time of the composite automaton is the minimum among the delay times of all individual automata. However, in the new state of the composite automaton each individual automaton will switch to a new "intermediate" state, which will record the remaining delay time of that individual machine.

Definition 3. (Formal definition is omitted.)

One specific kind of wiring is of particular importance, and that is the wiring of two machines into an *informatic pair.*

Definition 4. An *informatic pair* is a pair of automata wired together so that every input of one automaton is wired to an output of the other machine, and vice versa.

By definition, therefore, an informatic pair induces a closed automaton, whose trace is well-defined.

3. Formal Definitions of Intuitive IRTPS Concepts

We propose the model outlined in the previous section as one in which to couch discussion of IRTPS issues. Specifically, we propose viewing the IRTPS system as a machine wired to the environment. The various considerations that have been raised in the past can be given precise meanings in terms of the class of environment-machines being considered, their internal structure (which the "intelligent" system will be able to exploit), their wiring to and from the IRTPS (assumptions about input and output), and the internal structure of the IRTPS itself.

We do not propose a comprehensive list of criteria for IRTPS systems, only the setting in which to define them. We hope to reach a concensus of sorts on the criteria during the workshop itself. The most general formulation of them presumably is a some maximization of a utility function over time. Such a requirement would have to specify the class of environments being considered, and the utility function. This is sufficiently general that it's probably both correct and useless (though others may wish to correct us). What we will do instead is pick a few keywords that have been used in the past, and attempt to give them a more-or-less precise meaning in our model. In most cases, these definitions fall short of fully capturing the intuitions behind the terms;

we hope to improve these definitions as well as formulate new ones for other important terms at the workshop. Nonetheless, even these partial definitions contribute to our understanding of these terms by showing that some of them are neutral specifications of the performance required of an IRTPS system, while others are specifications of the internal workings of a particular IRTPS system architecture. (See related discussion in the workshop paper "Notes on Evaluating Methodologies ...")

Timeliness

Intuitively we require that an IRTPS system not only produce correct or useful output, but to do so at the right time. This requirement translates in our framework to a restriction on delays allowable between certain outputs of the environment-machine and its reaching certain other states later. In the simple form the restrictions are absolute, and in the more sophisticated form they are stochastic, allowing occasional harmful long delays in exchange for extra speed at most other times. (From here on we'll talk only of absolute requirements, understanding that the definitions can be extended to the statistical case.)

Some of the other notions below are mentioned in service of timeliness.

Internal Clock

Intuitively, this is a requirement that the IRTPS have a notion of the passage of time in the real world. In our framework this translates to a requirement that some component machine of the IRTPS change at a constant rate.

Recency

The IRTPS should not fall behind real time to handle a backlog of inputs and it should not operate on seriously out of date inputs--unless it has explicitly decided to do so. In terms of our framework, the probability of an environmental state change influencing an IRTPS system state change should decrease rapidly to 0 over time. If the IRTPS sytem has internal structure, then the probability that state changes in its components will affect one another should decrease rapidly to 0 over time.

Guaranteed Cycle Time

Intuitively, we require that the IRTPS never get "lost in thought" for an unbounded period. In our framework this translates to upper bounds on the state and input delays (*deltaS* and *deltaI*) of the IRTPS-machine.

Unpredictability

Intuitively, there is a significant number of environment state and output changes that the IRTPS system cannot predict. In terms of the framework, this means the rate of environmental state transitions varies widely.

Asynchrony

Given this unpredictability, an IRTPS system must receive inputs when the environment produces them and not on any arbitrary schedule. To make sense of this in terms of our framework, we need to model the IRTPS system as a wired set of sub-automata, including some for input reception. Then, the rate of state changes in input reception components must be independent of the rate of state changes in other components.

Data Glut

Intuitively, the IRTPS system will be overwhelmed with data, and will be able to act on only a small part of it. In our framework means that the wiring between the IRTPS system and the environment be dense, with many more input changes to the IRTPS machine than state changes at any given period.

Selectivity, Intelligent Data Filtering

Intuitively, in order to cope with this data glut, the IRTPS cleverly chooses which inputs to process. Defining intelligent data filtering in terms of our framework will depend upon the internal structuring of the machine as a set of wired sub-automata and specification of dependencies among their state changes.

Uncertainty, Noise

Intuitively, the data and world model available to the IRTPS system will be partial, approximate, and sometimes plain wrong. In our framework this translates into imperfect correlations between inputs to

and states of the IRTPS, on the one hand, and states of the environment, on the other hand.

Modeling, Prediction, Foresight

Under some architectures, the IRTPS system will have a model of the enviroment, which it can use to predict the future and thus guide its actions. In our framework this is captured by a requirement that some component of the IRTPS system have an internal structure that stands in correlation to the internal structure of the environment. Furthermore, this component must influence state transitions of the IRTPS.

Robustness, Graceful Degradation

Intuitively, as enviroments get more demanding, the performance of the IRTPS system should deteriorate only gradually. To define this concept in our framework, we would have to define measurements on demandingness of the environment and quality of performance of the IRTPS. For example, demandingness of the environment might be measured in terms of number of state variables or rate of state changes. Quality of performance would depend upon application-specific utility functions. Then graceful degradation implies that small changes in demandingness should produce small changes in quality.

4. Perspective on Real-Time Intelligent Agents

We have attempted so far to use as neutral a language as possible, free from terms for which different researchers have conflicting intuitions, or from presuppositions about possible solutions. We now deviate from this restriction and talk briefly about our perspective on real-time intelligent agents.

The term agents is used so much nowadays that it has become meaningless without reference to some particular notion of agenthood. What we mean by this term is a system that is embedded in a temporal framework, and about which one can talk in terms of its having knowledge and beliefs, desires and goals, reasoning capabilities, resource limitations, and similar other mentalistic-sounding terms.

We see several related advatages to dragging this notion of agenthood into the context of IRTPS. Among other things, it allows us to:

* ascribe to the system knowledge of the environment and the related abilities to interpret past events and to predict and plan for future events;

* talk of the goal the system has at any point in time, and the knowledge it needs in order to achieve the goal;

* design and explain the IRTPS as making tradeoffs among different goals, having knowledge of its reasoning resources, and making decisions such as whether to spend time incorporating new data into its world model.

In general, the agent level gives formal meaning to the "I" in IRTPS. If in addition we view the environment as containing other agents we have additional benefits. For example, we can:

* talk of the IRTPS having knowledge of the goals, knowledge and capabilities of these other agents;

* specify the class of environment in which the IRTPS is to function at least partially in terms of what agents it contains, whether they are hostile or friendly, what their capabilities are, etc.

Thus, "agent" is a useful concept to have, both in specifying the problem and (especially) in specifying architectures.

In order to make engineering sense of this concept, however, we must do at least the following:

1. Define the notions we associate with agents, such as knowledge and goals, including their temporal dimension. For example, what does it mean for a process controller to "know" that it "needs" to lower the pressure in the chamber within five seconds? What does it mean for the agent to "believe" that doing so will prevent an explosion?

2. Explain the connection between these high-level notions and low-level behaviors like sensing a signal or actuating an effector.

3. Explain the special constraints placed on agents by real-time environments.

We have both made some progress in this direction, which we describe briefly in two attachments to this paper. "Agent-Oriented Programming"

outlines a general approach to treating information systems as formal agents, and some related work that is taking place in Stanford's Robotics Lab. "Research on Adaptive Intelligent Systems" outlines architectural work and related applications being developed in Stanford's Knowledge System's Laboratory.

5. Recommendations regarding Testbed

5.1 Desirable Features of Testbed Environments

Testbeds should include simulated IRTPS environments both for development purposes and for controlled experiments. For verification, it will be necessary to test at least some scenarios in real environments, for example involving interface to a real vision module or to a real process controller.

Testbed simulations should provide appropriate sensors and effectors for use by application systems. They should be factorable--provide "black box" solutions to component tasks--so that researchers can choose to address only those parts of the application problem that interest them. Simulations should be tunable--permit constant factor speed modulations--so that researchers can ignore conventional issues of efficiency and concentrate on more fundamental research issues. Finally, simulations should be instrumented--equipped with meters on important performance variables--so that researchers can analyze the different aspects of performance of their application systems.

Testbeds also should provide a suite of modular application scenarios. Each scenario should be issue-oriented, stressing a particular aspect of intelligent real-time problem solving, such as hard real-time constraints, conflicting sensor data, need for synchronization, etc. Each scenario should include a simulation controller to "play" the scenario and an associated knowledge base (in some neutral representation scheme) for use by application systems in handling the scenario. There should be several variations on each scenario. Base scenarios are for use by researchers in developing aspects of their application system to address the scenario issue. Standard test scenarios should exhibit the same issue-related phenomena as the base scenario, but differ from it on incidental features. For example, a standard test scenario for hard real-time constraints might different from the base scenario on which event carries the constraint, when it occurs in the scenario, what other events occur in the scenario, etc. Standard test scenarios will allow researchers to determine whether they have developed problem-independent approaches.

Stress test scenarios also should correspond to the development scenarios, but they should exhibit extreme cases of the issue-related phenomena. For example, a stress test scenario for hard real-time constraints might impose much shorter time constraints or introduce many more competing events. These scenarios will allow researchers to determine how their approaches degrade under extreme conditions. These different types of scenarios merely illustrate the kinds of scenarios we think would benefit research in an IRTPS testbed.

5.2 Desirable Features of Testbed Domains

An IRTPS testbed domain should exhibit the general task characteristics identified in section 3. To take a few examples, we are interested in domains in which:

* it is not feasible to exhaustively sense interesting features of the environment;

* situation assessment requires interpretation of sensed data and fusion of data from multiple sensors;

* it is not feasible to enumerate every condition the IRTPS will encounter;

* the environment is orderly enough to permit probabilistic prediction of future events;

* coordinated courses of action are sometimes superior to sequences of locally determined actions;

* events vary in the deadlines associated with effective responses;

* multiple goals vary in importance;

* a broad range of relevant knowledge is available;

* explanations of phenomena and rationales for behavior are required.

Because different application domains may differentially emphasize particular subsets of these features, it would be preferable to identify at least two or three testbed applications to avoid artificially skewing research activities toward an arbitrary subset of relevant issues. Moreover, we do not believe that an IRTPS testbed application should replace other applications being studied in the research community, but

rather that a diversity of applications promotes a more complete exploration of the space of relevant research issues and solutions.

6. Recommendations for Structuring the IRTPS Research Program

Three programmatic objectives motivate our remarks on structuring the IRTPS Research Program:

* to stimulate new IRTPS research by providing environments in which new investigators could begin to study IRTPS issues without the overhead entailed in developing one's own application;

* to facilitate interactions and exchange of results in the IRTPS community by providing a communications medium in which investigators could demonstrate approaches originally developed in diverse application domains that are unfamiliar to their colleagues;

* to permit a more scientific approach to IRTPS research by providing a controlled environment for comparative evaluation of competing approaches.

As discussed at the first AFOSR workshop on IRTPS, development of a high-quality and easily accessible testbed application that meets all of the requirements would be an expensive and time-consuming task. If we aim for a diversity of testbed applications, as recommended in this paper, the cost rises proportionately. At the same time, this effort would be redundant with efforts already underway by individual researchers to develop a variety of interesting IRTPS testbed applications for their own work, for example:

* Hayes-Roth's simulation of the intensive care environment;

* Hayes-Roth's simulation of GaAs crystal growth by MBE;

* Cohen's simulation of fire-fighting in Yellowstone Park;

* D'Ambrosio's simulation of wilderness exploration;

* Lesser's simulation of vehicle fleet maneuvers;

* SRI's Flakey the robot;

* Latombe's Gofer robots.

Given AFOSR's budget and timetable for this research program, developing a new community testbed application does not seem to be the best use of scarce resources at this time.

As an alternative, we recommend that AFOSR use its IRTPS Research Program to encourage empirical research on testbed applications currently being developed in the community, with particular interest given to two kinds of proposals:

(a) Proposals that offer, in addition to new empirical research, to deliver a testbed version of their application domain suitable for use by other members of the community;

(b) Proposals that offer to comparatively evaluate alternative architectures through controlled experiments within their application domain.

This structuring of the program would address all three progammatic objectives:

* Testbed applications obtained under the first kind of proposal would stimulate new IRTPS research by providing environments in which new investigators could begin to study IRTPS issues.

* These testbed applications would facilitate interactions and exchange of results in the IRTPS community by providing a communications medium in which investigators could demonstrate different approaches.

* Research conducted under the second kind of proposal would specifically include comparative evaluation of competing approaches in a controlled environment.

This structuring of the program would offer two additional advantages:

* It would offer individual researchers more latitude in choosing an application domain for their work.

* It would guarantee exploration of a diverse set of application domains and a broad range of IRTPS issues within the research community.

IV. Intelligent Real-Time Problem Solving Workshop -- Dean

(This document was submitted to the workshop coordinators as an indication of research interest and willingness to participate in the workshop. It is included here because of its succint description of the IRTPS problems.)

Thomas Dean
Department of Computer Science
Brown University
Box 1910, Providence, RI 02912
Phone: (401) 863-7645
Fax: (401) 863-7657
Email: tld@cs.brown.edu

1 Rationale and Level of Participation

I have been doing research in the area of real-time planning and problem solving for the last three years. My students and I have published papers on the subject in IJCAI-89, AAAI-88, and several other conferences and workshops. I am working on a book entitled "Planning, Execution, and Control" which treats the subject of real-time planning and control in some detail, and, hence, I am very interested in tracking developments in the area. Contingent on the level of effort involved, I would be willing to act as a reviewer for one of the three exploratory groups and either lead or participate in a discussion of that group's work. The rest of this document consists of paragraphs drawn from recent papers and proposals arranged in some semblance of a coherent account of my IRTPS research interests.

2 IRTPS-Related Research Interests

The traditional view of planning in robotics and artificial intelligence as off-line computation relying on precise models and perfect information has been challenged by recent work on *situated activity*: the study of robotic control systems embedded in complex environments.

The challenge has been met with proposals for *reactive systems*: systems designed to react directly to perceived conditions in situations where there is little or no time to deliberate on how best to respond. The work on reactive systems, however, has diverted effort from *planning*: predicting possible futures and formulating plans of action that take into account those possibilities. My current research can be seen as attempting to connect traditional research in planning with the constraints governing embedded systems, by reformulating the process of planning in terms of control. Viewed from a control perspective, reactive systems embody particular strategies for controlling processes. In order to evaluate reactive systems, we must analyze the connection between such strategies and the physical systems they seek to control. The tools required to perform such analyses are readily available from computer science, control theory, artificial intelligence, and the decision sciences.

Over the last few years, my students and I have examined a number of the tradeoffs involved in building robotic systems that solve time-dependent control problems. We assume that robotics engineers have as their primary goal the design of high performance systems. One can often improve the performance of a robotic system by trading accuracy for speed in decision making. In some cases, the tradeoffs can be made at design time and compiled into a run-time system. Alternatively, general knowledge for making the tradeoffs can be built into the system, enabling it to make the necessary tradeoffs at run time. Many systems will employ a hybrid strategy involving compiled design-time tradeoffs coupled with some means of making additional run-time tradeoffs.

Previous approaches to planning have taken a rather cavalier attitude toward the availability of computational resources. The fact that planning systems have limited computational resources leads to the need for a theory of time-dependent planning. In our framework for time-dependent planning, we view decision processes as consumers of computational resources, and producers, as well as consumers, of informational resources. We refer to the task of allocating computational resources to decision processes to maximize some measure of utility as *deliberation scheduling*, and we employ standard methods from decision theory to analyze time-dependent planning problems and their solutions.

By making use of a class of approximate decision procedures we call *anytime* algorithms,[1]

[1] An anytime algorithm corresponds to a decision procedure that can be interrupted at any time during its execution to return an answer. For the sorts of anytime algorithms we are interested in, one can determine the expected utility of the answers returned by the corresponding decision procedure as a function of the time spent in computing the answers. The most useful anytime algorithms are those whose expected utility

we are able to view the problem of time-dependent control as a combinatorial optimization problem and analyze it in terms of the theory of computation. For certain control problems, we have been able to devise strategies for making tradeoffs at run time that are provably optimal. We are also looking at design-time strategies for generating optimal run-time systems. By casting a time-dependent control problem as a sequential decision-making problem, we can apply stochastic modeling techniques and the theory of Markovian decision problems to analyze the tradeoffs involved in using a fixed-computation-time decision procedure as part of a run-time system.

We view planning as the process of anticipating possible future courses of events so as to choose among them. Planners are embedded systems that have to synchronize their behavior with that of the world in which they are embedded. Other researchers noting the expressive limitations of the predictive models used in early planning research have sought to develop better models that capture uncertainty and ignorance on the part of the planning systems. In particular, researchers have begun to explore various stochastic modeling methods [17, 20, 21, 1] that make more realistic assumptions about what an agent might know about its environment. We have developed one such model for prediction [7, 8] that extends previous work in temporal reasoning and promises to provide a basis for planning under uncertainty. Unfortunately, the inference required for computing our stochastic temporal models is computationally prohibitive [5], and, hence, we fall prey to the complaints of those researchers who have abandoned research in planning on the basis that it makes unrealistic assumptions about available computational resources. Recently, however, several researchers have devised methods [4, 14] for computing approximations in a manner consistent with our approach to time dependent planning [6, 2]. It is our expectation that such approximation algorithms coupled with methods of Horvitz [11, 12, 13], Russell and Wefald [19], Hansson and Mayer [9] and others will pave the way to developing a realistic model of planning under uncertainty and time constraints.

Ultimately, we would like some means of synthesizing high-performance run-time systems from a model of the anticipated run-time environment and a set of performance criteria [15, 18, 16, 10]. Unfortunately, the current methods for automated synthesis using existing modeling techniques are prohibitively expensive from a computational perspective. In order to reduce the complexity of synthesis, we are seeking alternative modeling techniques that employ only a small number of parameters. For instance, by characterizing the utility of the

---

increases monotonically over some significant range of computing times.

components of a run-time system as linear functions of a few parameters, certain synthesis problems reduce to solving a set of simultaneous equations. While the alternative modeling techniques often fail to capture important features of the run-time environment, they do appear to be useful for certain classes of time-dependent control problems; characterizing this class precisely is one of our current objectives.

References

[1] Agogino, A. M., Srinivas, S., and Schneider, K., Multiple Sensor Expert System for Diagnostic Reasoning, Monitoring, and Control of Mechanical Systems, *Mechanical Systems and Signal Processing*, (1988).

[2] Boddy, Mark and Dean, Thomas, Solving Time-Dependent Planning Problems, *Proceedings IJCAI 11, Detroit, Michigan*, IJCAI, 1989, 979–984.

[3] Brachman, Ronald J., Levesque, Hector J., and Reiter, Raymond, (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, (Morgan-Kaufman, Los Altos, California, 1989).

[4] Chavez, R. Martin, *Fully Polynomial Randomized Approximation Schemes for the Bayesian Infererencing Problem*, Report KSL-88-72, Section on Medical Informatics, Stanford University School of Medicine, 1988.

[5] Cooper, Gregory F., *Probabilistic Inference Using Belief Networks is NP-hard*, Technical Report KSL-87-27, Stanford Knowledge Systems Laboratory, 1987.

[6] Dean, Thomas and Boddy, Mark, An Analysis of Time-Dependent Planning, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 49–54.

[7] Dean, Thomas and Kanazawa, Keiji, Probabilistic Temporal Reasoning, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 524–528.

[8] Dean, Thomas and Kanazawa, Keiji, Persistence and Probabilistic Inference, *IEEE Transactions on Systems, Man, and Cybernetics*, 19(3) (1989) 574–585.

[9] Hansson, Othar and Mayer, Andrew, The Optimality of Satisficing Solutions, *Proceedings of the 1988 Workshop on Uncertainty in Artificial Intelligence, Minneapolis, MN*, 1988.

[10] Heckerman, David E., Breese, John S., and Horvitz, Eric J., The Compilation of Decision Models, *UW89, Windsor, Ontario*, 1989, 162–173.

[11] Horvitz, Eric J., Reasoning About Beliefs and Actions Under Computational Resource Constraints, *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence, Seattle, Washington*, 1987.

[12] Horvitz, Eric J., Reasoning Under Varying and Uncertain Resource Constraints, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 111–116.

[13] Horvitz, Eric J., Cooper, Gregory F., and Heckerman, David E., Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study, *Proceedings IJCAI 11, Detroit, Michigan*, IJCAI, 1989, 1121–1127.

[14] Horvitz, Eric J., Suermondt, H. Jacques, and Cooper, Gregory F., *Bounded Conditioning: Flexible Inference for Decisions Under Scarce Resources*, Technical Report KSL-89-42, Stanford Knowledge Systems Laboratory, 1989.

[15] Kaelbling, Leslie Pack, Goals as Parallel Program Specifications, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 60–65.

[16] Kanazawa, Keiji and Dean, Thomas, A Model for Projection and Action, *Proceedings IJCAI 11, Detroit, Michigan*, IJCAI, 1989, 985–990.

[17] Pearl, Judea, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, (Morgan-Kaufman, Los Altos, California, 1988).

[18] Rosenschein, Stan, Synthesizing Information Tracking Automata from Environment Descriptions, In Brachman et al. [3], 386–393.

[19] Russell, Stuart J. and Wefald, Eric H., Principles of Metareasoning, In Brachman et al. [3].

[20] Shachter, Ross D., Evaluating Influence Diagrams, *Operations Research*, 34(6) (1986) 871–882.

[21] Wellman, Michael P., *Formulation of Tradeoffs in Planning Under Uncertainty*, Technical Report MIT/LCS/TR-427, MIT AI Laboratory, 1988.

V. Real-Time Problem Solving: Preliminary Thoughts

N. S. Sridharan, R. T. Dodhiawala
FMC Corporate Technology Center, Santa Clara, CA

Abstract

Before real-time problem solving approaches gain widespread fascination leading a flurry of half-hearted attempts, it is important to review the range of issues involved to successfully build such problem solvers. We are at an appropriate juncture in the work on real-time problem solvers to be able to highlight the basic issues. This paper presents preliminary thoughts on the nature of the real-time problem and the issues involved in building such systems - ranging from architectures to performance evaluation and feedback. There are a few issues, like verification and fault-tolerance, that are not discussed.

1. Model of a Real-Time Problem Solver

A model of a distributed, real-time problem solver is shown in figure 1-1.

The main features of the model are:

- The system is event-based: all interactions between the system and the environment and between subsystems is captured in events. Processing in the system is driven by the need to respond to these events. Events must specify deadlines by which they may be responded to.
- The system has buffers at various interfaces, permitting asynchronous operation.
- The system interacts with the environment via sensors and effectors. Sensors collect and buffer data from the environment. Effectors accept signals which represent commands (for example, on/off, set-speed) that have been buffered by the system. Pre- and post-processing of the sensor data and effector commands may be necessary.
- The system interacts with one or more operators in an asynchronous fashion. The interaction is facilitated by display devices and graphical interfaces.
- The system performs assessments and maintains a model of the environment. Task models, domain goals, and performance goals enable replanning according to the established policies for reaction.
- The meta-level system performs the planning and scheduling of domain actions by considering deadlines and required response(s) using the resource model. The resource model provides a description of the use and availability of the physical resources (like sensors) as well as computing resources.

We use this model of the system to define terms, to explore issues and approaches to real-time problem solving. The type of problems we are interested in cover a spectrum that can be categorized as follows:

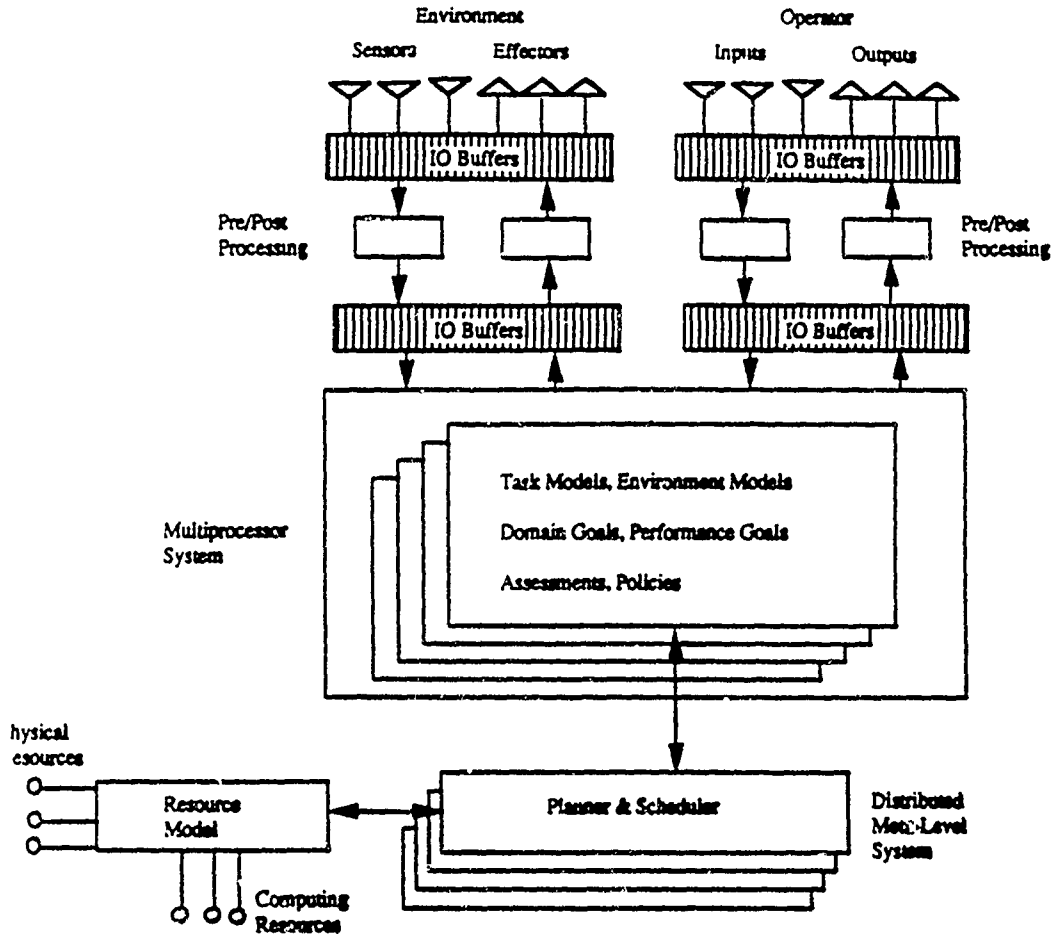


Figure 1-1: Model of a Real-Time Problem Solver

- simple, non-interactive, expert systems (example, for online diagnostics of automobile functions)
- telerobotic systems where the major decision-making responsibility rests on the operator (example, ground controlled space probes)
- multi-agent systems working in controlled environments with limited problem-solving capability (example, multiple autonomous munition handling robots on a carrier deck)
- interactive, knowledge-based systems operating in highly complex environments (example, Pilot's Associate, Submarine Operations Automation System).

Different real-time issues become prominent as we progress through the problems from the simple to the complex, and there is a need for common terminology with which to describe the

various aspects of real-time performance.

1.1. Terminology and Definitions

Speed alone is not real-time. A knowledge processing system must address the following aspects of real-time performance: *speed, responsiveness, timeliness*, and *graceful adaptation*. We define these four aspects of real-time next.

Speed is defined as the number of tasks executed per unit time. Typically, this is a function of the type of tasks, the hardware/software systems being used, and the granularity of the tasks.

Responsiveness is defined as the ability to take on new tasks, that is, responding quickly to user interaction and to stay alert to external data. Responsiveness facilitates reactivity by allowing the system to focus attention on the more critical data for further processing.

Timeliness is defined as the ability of the system to react to and meet deadlines. Predicting response times is an aspect of timely behavior, allowing the system to prefer more critical tasks based on the prediction.

Graceful adaptation is defined as the ability of the system to reset task priorities to adjust to changing workload or resources. Graceful adaptation allows for a smooth transition across potential perturbations to the system. Graceful adaptation also includes monitoring performance indicators and resetting system priorities to ensure performance within the desired bounds. Thus, graceful adaptation allows system behavior to adapt to demands of the external as well as internal destabilizing effects; the performance of the system must degrade gracefully rather than cause the system to halt abruptly.

While speed is indeed fundamental to real-time, a knowledge based system must exhibit all the four aspects for acceptable real-time performance.

1.2. Symptoms of Real-Time Failure

Performance failure of a real-time system can be characterized in several different ways. We can look at the *level* of the system at which a failure occurred, or base it on the *type of event* the system was attempting to respond to, or simply classify the failure along one or more observable *symptoms*.

A system may be described at three distinct *levels* as follows:

- Problem Level: This level consists of the environment, the operator, and the issues in the problem domain and the requirements on solving the problem. Failures here are indicative of the lack of real-time considerations at the problem formulation stage.
- Design Level: This level consists of the hardware and software design. This layer addresses the functional and performance requirements of the system, typically hosted on a conceptual architecture or problem-solving framework. Failures at this level are due to lack of appropriate technology (hardware, software and algorithms) to support real-time performance, the misapplication of the existing technology or the lack of performance models to evaluate system's performance.

- Implementation Level: This level is the actual physical system mounted on the appropriate hardware and software platform(s). The implementation consists of the application system, application support tools, software support, and hardware. Failures at this level include hardware malfunctions (poor reliability), poor quality control of the software, and inadequate performance probes and metrics.

The design level is the more interesting level to study real-time issues, because it can be independent of particular applications or implementation methodologies. While there are interesting real-time issues at the problem and implementation levels, our discussion will be directed primarily at the design level.

There are several dimensions of real-time failures. The severity of the failures depends on the stress on the system, but the following *failure symptoms* may be observed.

- Communication traffic jams: there is too much information that is being transmitted between environment and system or between the subsystems.
- Ignored critical events: the system is incapable of recognizing critical events from the non-critical events for further processing.
- Poor solution quality: the system does poorly in the trade-off between solution quality and time.
- Wrong solution: in an attempt to meet deadlines, the system's response is incorrect.
- No solution: the system is unable to produce even an acceptable partial solution in the given time.
- Data glut: the system is unable to manage incoming data effectively, causing a backlog of data at its input ports.
- Breakdown of communication with interactive user: the system is not responsive to the needs of the user with which it interacts.
- Thrashing over resource allocation: the system is being overly reactive and is unable to maintain its focus of attention on the critical events or tasks long enough without reallocating resources.

While this is not an exhaustive list, it does reflect on the kind of symptoms an intelligent real-time problem solver must address.

3. Factors of the Problem Environment Affecting Real-Time Performance

A real-time system is intended to operate in environments with the following characteristics:

- Dynamic: Most of the effects observed in the environment are beyond the control of the system and introduce large variations in the levels of priority and real-time stress. The effects of the system's planned actions are often unpredictable also, adding to the dynamics of the environment in which the system operates.
- Continuous Interactions: Even after being responded to, the changes are the basis of continuous interactions possibly because the influence of planned actions is often hard to comprehend over future time intervals.
- Uncertainty: The level of certainty of the information is generally low mainly due to

poor sensors or too much information with too little content needs to be assimilated in an unreasonable amount of time (data glut).

- Time-bounded Interactions: The response times are fairly short with stringent deadlines which if not met may cause cascading effects possibly rendering system to fail catastrophically.
- Discontinuous Changes: The changes may be abrupt and sudden, hard to anticipate and cause major perturbation leading to excessive demands on time and resources. The discontinuous changes tend to destabilize the system beyond recovery.

Not all these characteristics may be present in all real-time environments and various manifestations of these are also possible.

4. Approaches

The discussion in this section is intended to indicate that some thought has already been given to real-time problem solving but there is a need for developing sound techniques that will facilitate further advancement of the issues involved in successful building and deployment of such systems. Thus, we set the motivation for exploring the issues and approaches in this section.

4.1. Stages of Development of Real-Time Systems

Knowledge based systems are increasingly being used to address real-time problems that are too complex for conventional real-time approaches. Knowledge based systems, however, introduce some new issues while solving some old ones. The one that stands out prominently is the difficulty of predicting real-time response times. Knowledge based systems utilize search methods and pattern-matching techniques extensively, that make run time highly data-dependent and context-sensitive. In order to address the predictability and other such issues, we feel that real-time considerations must not be relegated simply to implementation time tactics of code tuning and ad hoc enhancements. They must be a part of the total design and development effort, and, in fact, spans the life cycle of the application.

Figure 4-1 shows the mapping of the various stages of development of a knowledge-based application to the real-time issues that are relevant at each of these stages.

Real-time performance cannot be achieved by run-time capabilities alone. Even the best approaches based on the good scientific principles may not work under particular situations and cannot be depended on for delivering the desired performance all the time. Real-time issues are pervasive at all stages of the development. The "insertion" of real-time solutions begins right at the problem formulation stage. Given an application, there may be ways to specify the problem such that it becomes amenable to real-time techniques downstream. Real-time considerations may then be given attention during design; knowledge engineering; problem-solving architecture development; performance testing, validation and tuning; and finally, dynamically at run-time. A certain amount of engineering is essential to meet the demands of today's applications. Generic approaches, while relevant, may just not work. A careful balance between these extremes of science and engineering is essential, both to meet performance requirements as well as to improve system robustness.

Our experience may be used for evaluating the IRTPS problem(s) and the proposed approaches

Stages of Development	Real-Time Issues
Problem Formulation	Real-Time Requirements
Conceptual Design	Tunable Architecture, Performance Model and Metrics
Knowledge Engineering and Prototyping	Real-Time Problem Solving Framework and Performance Measuring tools
Performance Tuning and Testing	Tuning the Architecture, Problem-Solving Framework, and Run-Time Control
Validation and Demonstration	Predictable Performance

Figure 4-1: Real-Time issues mapped to the stages of development of a problem solver

along these lines. We will also study canonical problem situations and generate and evaluate approaches that address the issues.

4.2. Framework for Discussing Approaches

We now present a framework, shown in figure 4-2, which will be used to discuss the approaches.

We must combine the best of conventional real-time approaches with AI techniques for effectively addressing real-time issues in a knowledge-based system.

Man-machine interface must consider bandwidth of communication between the operator(s) and the system, and how it changes according to the levels of autonomy.

Real-time issues in knowledge representation call for knowledge intensive meta-level control. Based on performance status, the system determines the impact on performance goals which if needed may be dynamically changed according to predefined policies.

Reasoning in real-time primarily influences solution quality versus effectiveness tradeoffs. The main issue is to produce acceptable solutions using limited resources. Performance models, temporal reasoning and resource calculus can be used to determine tradeoffs.

Problem formulation must consider alternative approaches to the problem with predictable tradeoffs in quality of the solutions versus the effort expended. Real-time issues must begin to

AI	• Man-Machine Interface	• Interactivity - Levels of Autonomy
	• Representation	• Meta-Level Needed - Performance Status - Performance Goals - Policies
	• Reasoning	• Resource-Limited Reasoning - Temporal Reasoning - Resource Calculus - Performance Models
	• Problem Formulation	• Alternative Approaches to Solution - Predicting Quality vs Effort
	• Problem-Solving Framework	• Tunable - Embedded (Implicit) Control - Solution Quality vs Performance Tradeoffs (Time, Resources)
	• Knowledge Engineering	• Real-Time Requirements Definition - Extend KE Methodology
	• Prototyping Methodology	• Real-Time Feasibility Analysis - Preliminary Evaluation of Real-Time Performance
	• Performance Measures and Metrics	• New Definition of Responsiveness, Timeliness, and Graceful Adaptation - Performance Probes and Indicators - Scheduling and Control over focus
Conventional	• System Architecture	• Tunable, Controllable, and Predictable
	• Design Methodology	• Early Performance Prediction on Prototype
	• Hardware	• Interruptability
	• Software	• Scheduling, Distribution, and Measures

Figure 4-2: A framework for investigating real-time approaches

play a prominent role starting at this stage.

The problem-solving framework must be tunable to the needs of the application. The main advantage of a problem-solving framework is to make selected control decisions implicit with the capability to reason about tradeoffs in time and resources through predefined control flows which match problem situations.

Knowledge engineering is a key step in the proper identification and acquiring of performance requirements. The knowledge engineering methodology needs to be extended to include real-time considerations according to the definition of real-time.

Prevalent rapid prototyping methodology must be extended to not only evaluate the functional but also the real-time performance feasibility of the approaches.

Performance measures and metrics are essential to validate real-time approaches. They must be based on the definition of real-time so that they can be controlled to ensure that the system performs within the desired bounds. Performance measures and metrics must be defined both for the architecture and the domain system.

System architecture must be tunable to permit experimentation with alternate approaches, be controllable and predictable to facilitate acceptance.

Design methodology must incorporate mechanisms which facilitate rapid prototyping and performance prediction.

Besides the conventional real-time needs for interrupts, etc., the hardware system must provide the capabilities for hosting the software and application systems efficiently.

Software system must support the scheduling, communication, distributed implementation, and collecting performance data.

4.3. Solution Candidates

Based on our experiences in the Real-Time Risk Reduction (RTRR) program (sponsored by DARPA, subcontract to the Pilot's Associate program) as well as work on a real-time process management application, we are well prepared to bring to the project a broad perspective on both the science and engineering aspects of developing real-time knowledge-based systems.

Our earlier work on real-time systems involved studying general real-time issues, investigating available and imminent technology (both conventional and AI, hardware, software and algorithms) that will support real-time performance, developing innovative solution candidates for the near term with potential for evolutionary growth, develop a real-time knowledge processing architecture and demonstrate its validity for tuning and achieving real-time performance, and developing and demonstrating a scenario from the Pilot's Associate domain.

An important part of our earlier work was developing performance measures and metrics that can be used to evaluate real-time performance. Timeliness used a metric called Event Response Timeliness (ERT) which was defined in terms of importance-weighted scores of average lateness for tasks that were late. Responsiveness used a metric Response Latency, which was defined in terms of average time delay between the time of arrival of an event in the buffer and the time at which the task was picked for execution. Extensive experimentation and performance evaluation was a key feature of the RT-1 work under RTRR. From our early work, we know that a system cannot show extremes of both responsiveness and timeliness at the same time. The nature of our work will involve developing such *fundamental insights* that are supportable empirically and explainable theoretically. We believe that studying similar limitations is crucial to developing the means to alleviate the impact of these limitations.

Our solution candidates capture essential aspects of approaches for real-time performance, and represent the best of stock of current ideas.

- Control Reasoning: Use knowledge about task demands, time constraints, goals and system resources to select methods and formulate schedules. ~ improves timeliness and graceful adaptation. The specific solutions under this category include supervisory control, temporal reasoning, discretionary IO, and speed/effectiveness

tradeoff.

- Focus of Attention: Use designs that permit quick and efficient shift of attention to the more critical events. It improves responsiveness and graceful adaptation. The specific solutions include interruptibility of knowledge sources, interruptibility of the top level control cycle, and prioritized triggering. Control reasoning is needed for successful application of focus of attention techniques.
- Parallelism: Execute knowledge sources in parallel and allow asynchronous event transmission. It improves timeliness and speed. The specific solutions include asynchronous execution of knowledge sources, asynchronous execution of the top level control cycle, parallel execution of knowledge sources.
- Algorithm Efficacy: Design new algorithms or redesign existing algorithms to deliver approximate and acceptable solutions quickly. It improves timeliness and graceful adaptation. The specific solution candidates include incremental algorithms, algorithmic caching, cascaded algorithms, and anticipatory processing.

5. Current and Future Work

We list most of the issues in real-time problem solving and the early ideas on the approaches on the basis on extensive study as well as implementation in the course of several applications at FMC. The need to define basic concepts is needed to come to terms with the issues. There have been several simultaneous attempts in the AI research community to define the basic concepts. While we have merely presented our perspective, our study shows that it is mostly consistent with other work. The two main areas where we direct our current efforts are architectures for real-time systems and performance metrics. Future work involves problem-solving architectures for classes of problems and verification of both functional and performance requirements.

VI. Agent Oriented Programming

Yoav Shoham
Computer Science Department
Stanford University

1 Introduction

This is an abbreviated version of a manuscript that describes work we are doing in the Artificial Agents group in the Robotics Lab. It touches on issues that are subject of much current research in AI, issues that include the relationship between a machine and its environment, and the notion of agenthood. Many of the ideas here intersect and interact with ideas of others. For the sake of continuity, however, I will delay placing this work in the context of other work until the end.

The term 'agents' is used a lot these days. This is true in AI, but also outside it, for example in connection with data bases and manufacturing automation. Although very popular, the term has been used in such diverse ways that it has become almost meaningless without reference to a particular notion of agenthood. Some notions are primarily intuitive, others quite formal. Some are very austere, defining an agent essentially as a Turing-like machine, and others ascribe to agents sensory-motor, epistemic and even natural language capabilities.

We propose viewing 'artificial agents' as formal versions of human agents, possessing formal versions of knowledge and beliefs, desires and goals, capabilities, and so on. The result is a computational framework which I will call *agent-oriented programming.*

The name is not accidental, as AOP can be viewed as an extension of the *object-oriented programming* (OOP) paradigm. I mean the latter in the spirit of Hewitt's original Actors formalism, rather than in the more technological sense in which it is used nowadays. Intuitively, whereas OOP proposes viewing a computational system as made up of modules that are able to communicate with one another and which have individual ways of handling

in-coming messages, AOP expands the picture by allowing the modules to possess knowledge and beliefs about one another, to have goals and desires, and other similar notions. A computation consists of these agents informing, requesting, negotiating, competing and assisting one another.

This is the programming-paradigm perspective on AOP. An alternative view of AOP is as a formal language. From this perspective it may be viewed as a generalization of epistemic logics, which have been used a fair amount in AI and distributed computation in recent years. These logics describe the behavior of machines in terms of notions such as knowledge and belief. These mentalistic-sounding notions are actually given very precise computational meanings, and are used not only to prove properties of distributed systems, but to design them as well. A typical rule in such a 'knowledge-based' system is "if processor A does not know that processor B has received his message, then processor A will not send another message." AOP expands these logics by augmenting them with formal notions of goals, desires, capabilities, and possibly others. A typical rule in the resulting framework would be "if agent A knows that agent B intends to do something agent A does not want done, A will request that B change his intention." In addition, temporal information is included to anchor knowledge, desires and so on in particular points in time.

Intentional terms such as knowledge, beliefs, goals and so on are used in a curious sense in the formal AI community. On the one hand, the definitions (e.g., of knowledge) come nowhere close to capturing the full linguistic meanings. On the other hand, the intuitions about these formal notions do indeed derive from the everyday, commonsense meanings of the words. What is curious is that, despite the disparity, the everyday intuition has proven a good guide to employing the formal notions, when done in moderation.

The rest of this document is organized as follows. We first provide motivation for the Agent-Oriented Programming paradigm by looking at three futuristic applications. We then outline the research programme, including a sketch of the progress we have made to date. We conclude with a brief comparison with recent related research efforts.

2 Agent-Oriented Programming in the year 199x

We present three futuristic scenarios. The first involves only human agents, the second both humans and machines, and the third only robots.

2.1 Office automation

(omitted)

2.2 Manufacturing automation

Alfred and Brenda work at a car manufacturing plant. Alfred handles regular-order cars, and Brenda handles special-order ones. The plant has a welding robot, known affectionately as Calvin. The plant foreman is Dashiel. The following scenario develops.

At 8:00 Alfred requests that Calvin promise to weld ten bodies for him that day. Calvin agrees to do so. At 8:30 Alfred requests that Calvin accept the first body, Alfred agrees, and the first body arrives. Calvin starts welding it and promises to let Calvin know when it is ready for the next body. At 8:45 Brenda requests that Calvin work on a special-order car which is needed urgently. Calvin responds that it can't right then, but that it will when it finishes its current job, at approximately 9:00. At 9:05 Calvin completes welding Alfred's first car, ships it out, and offers Brenda to weld her car. Brenda ships it the car, and Calvin starts the welding. At 9:15 Alfred enquires why Calvin is not yet ready for his (Alfred's) next car. Calvin tells him why, and also that it (Calvin) expects to be ready by about 10:00. At 10:05 Calvin completes the welding of Brenda's car, and ships it out. Brenda requests that he reaccept it and do some painting, but Calvin refuses, explaining that it doesn't do painting. It then offers to weld another car for Alfred. Calvin then proceeds to weld Alfred's cars, until 12:15 at which time Brenda requests that Calvin commit to welding four more special-order cars that day. Calvin replies that it cannot, since there are still six more cars it promised to weld for Alfred that day. Brenda requests from Alfred that he allow Calvin to work on her cars first, finishing as many of Alfred's cars as it can afterwards. Alfred refuses. Brenda requests that Dashiel order Calvin to

accept her Important Request and cancel its commitment to Alfred. Dashiel orders Calvin to weld two of Brenda's cars, and then as many of Alfred's as time allows.

2.3 Gofer robots

It is 1995, and the new Stanford Information Sciences building has been completed. In addition to its human inhabitants, the building is populated by about 100 Gofer robots. The role of the Gofers is to carry documents to and from the copying machine, fetch coffee and sodas, and generally make themselves useful.[1]

At a junction of corridors two Gofers, G-Ed and G-John, engage in the following exchange.

G-Ed: I intend to turn into the north corridor.

G-John: So do I; you may go first. Where are you headed?

G-Ed: The copying machine.

G-John: In that case, will you xerox this document for me and drop it off at John's office?

G-Ed: Ok.

G-Ed proceeds down the north corridor, and G-John scuttles back down the west corridor. Half way down the corridor it runs into G-Nils.

G-Nils: Where are you headed?

G-John: To John's office.

G-Nils: Watch out, there's a traffic jam in corridor #25; G-Terry and G-Mike collided, and it's not a pretty sight.

[1]The presence of the Gofers is an actual plan; the Gofer project in out Robotics Lab is currently experimenting with three platforms. The completion of the building by the said date, however, is pure speculation.

3 Research programme

The preceding scenarios made reference to mentalistic notions such as knowledge, belief, desires, goals and capabilities. The goal (...) of our research is to make engineering sense out of these abstract concepts. The result is to be a programming paradigm which we call Agent-Oriented Programming. This framework is to have three primary components.

- A restricted formal language of intentional constructs such as beliefs and goals, with clear syntax and semantics. This language will be used to define agents.
- A programming language in which to program these agents, with primitive commands such as request and offer. The semantics of the programming language will be derived from the semantics of agents.
- A compiler from the agent-level language to a machine-level language.

In the following subsections we expand on these components a little bit.

3.1 Language definition

We need to first define a precise language for talking directly about knowledge, beliefs, goals, and similar properties of agents. We have already begun work in this direction; here are a few examples of statements in the language of agents.

- The fact that at 9:00 Bob has a goal to purchase X-terminals at 10:00 is expressed by

$$\langle 9:00, G_{Bob}\langle 10:00, purchase(Bob, Xterminals)\rangle\rangle$$

- The fact that at 9:15 Alfred believes that at that time Calvin has the goal of welding car #14 at 10:00 is expressed by

$$\langle 9:15, B_{Alfred}\langle 9:15, G_{Calvin}\langle 10:00, weld(Calvin, car\#14)\rangle\rangle\rangle$$

- The fact that on Monday G-John knows that on Thursday he will know whether on Wednesday G-Ed copied document #114 on Wednesday is expressed by $\langle Monday, K_{G-John}\langle Thursday,$
$(K_{G\,john}\langle Wednesday, copy(G-Ed, document\#114)\rangle \vee$
$K_{G\,john}\langle Wednesday, \neg copy(G-Ed, document\#114)\rangle)\rangle\rangle$

To define the language precisely its syntax and semantics must be fixed. Appendix A provides a few more details of the syntax, in its present stage of development. A fuller description of both syntax and semantics can be found in an article authored by Becky Thomas, Sarit Kraus and myself.

3.2 A programming language

The language discussed above will define the concept of artificial agents. The second step will be a development of a language in which to program these agents. Basic operations in this language will include:

- execute a primitive action;
- inform;
- request;
- consent;
- offer;
- promise;
- persuade;
- negotiate

Both preconditions and effects of these actions will refer to goals, beliefs and so on. For example, in its simplest version, a precondition of informing is that the speaker knows the information, and post conditions are that the hearer knows it, that the speaker knows that the hearer knows, and so on.

This illustrates also the extent to which we will deviate from common sense. The act of informing among human beings is very complex, and involves many considerations, such as the speaker believing that the hearer

does not already know the information, the fallibility of human knowledge, and so on. We will incorporate into the formal language just as many of those features as are both needed and amenable to formalization.

As is well known from speech-act theory, the interpretation of communicative acts can be complex. For example, an apparent informing act may actually serve as a request ('that sandwich looks good'). We may wish to incorporate some of these properties into the programming language.

As of now we have no report on the design and implementation of this programming language.

3.3 Compiling into the language of machines

We have said that we intend to view machines as well as agents. If we wish to do that, however, we face the problem of bridging the gap between the AOP level and the agentless language of machines. Consider for example the task of coordinating a welding robot with other activities in the plant. We may find it convenient to speak as if the robot has "knowledge" of the cars waiting to be welded and a "goal" to weld one of them, but ultimately we need to connect to the sensors on board the welder and to its controllers.

For that we need to "compile" statements in the AOP to the machine language, so that, for example, the high-level command "offer to weld the next car" will be translated to the appropriate control commands. Of course, each machine will have highly idiosyncratic sensors and controls, and so the target language of our compiler will have to be at a slightly higher level than the individual machine. We model machines more abstractly as consisting of an internal state, input and output, and a transition function with an associated time delay. These machines can be aggregated, yielding quite complex behaviors. The compiler will take AOP expressions and commands, and translate them into input to those machines.

We propose to use the machine language outlined in our report with Barbara Hayes-Roth as our target language.

We have not yet tackled the compilation process head on. The closest we have gotten to doing so is to look at a particular aspect of the transition from high-level commands to low-level controls, namely its tolerance to small perturbations: the high level command can be reduced to more than one low level command. This comes up whether the high-level command includes intentional operators or not. For example, it is unreasonable to interpret the

gross command 'roll down the middle of the corridor' as specifying a precise geometrical trajectory. Instead, we propose viewing it as a *protogram*. A protogram specifies the prototypical behavior, an ideal that is deviated from due to the interaction with other protograms (such as the 'avoid obstacles' protogram). Our current thinking about protograms is described in an appendix in the full manuscript.

4 Brief comparison with other work

Much work has been directed at defining agents, machines and environments. Some of the work is similar to ours. In fact, some aspects of this research programme were inspired by this previous work. Other work is quite different, even though it uses similar terminology. Here I briefly mention the relationship I see to several projects in and around Stanford, given my limited knowledge of them.

- McCarthy's ELEPHANT language. Appears similar with respect to the vision of a programming language, its primitive commands, and the general wish to endow them with formal semantics.

- Winograd's project on coordination. I don't know enough to really say yet; my initial perception is that our intuitions overlap significantly, but that Winograd does not intend to rest his system on formal foundations.

- Nilsson's work on Action Nets. Similar in the desire to incorporate intentional notions into the machine model (actually, talks only of beliefs and goals). Significantly different machine model, and very different way of incorporating the intentional notions. Similar emphasis on the real-time nature of the machine.

- Genesereth's work on agents. Genesereth's model of machines inspired the one defined here, although some differences exist. In his work these machines are called agents, whereas we associate the term with the intentional level.

- Barbara Hayes-Roth's work on agents. Is similar in its general goal to combine high-level, cognitive-like behavior with real-world input and output. Emphasizes less the theoretical framework and the inter-agent

aspects, and more the experimental methodology and structure of the individual agent.

- Rosenschein and Kaelbling's work on situated automata. Probably the strongest influence on our work. Differences: different intensional languages (they have no time and only the K operator, which makes reasoning at the intentional level less interesting), and also a slightly different machine-level language (no real-valued delays, as far as I can tell).

- Work, mostly at SRI, on belief, desire and intention (by Cohen, Levesque, Pollack, Konolige, Moore and others). Similar in motivation to ours as far as the semantics of agents goes. Cohen and Levesque's definition of goals is similar to (and preceded) ours, though we find that our explicit temporal framework is easier to use than the dynamic-logical language they adopt.

These are only a few of the related projects. There are others, both around Stanford and farther away. Rumor has it that some work is happening even in Massachusetts. A more detailed comparison will be good.

A A sketch of the agents language

Our language takes the notion of time as basic. Our most basic well-formed formulas (wffs) have the form

$$\langle t, p \rangle$$

where t is a time point and p is a proposition. This means that proposition p is true at time t.

If φ and ψ are wffs, then so are

- $\varphi \wedge \psi$, meaning *varphi* and ψ

- $\neg\varphi$, meaning not φ

- $\forall t\, \varphi$ (where t is a variable standing for a time point), meaning that φ is true at all times t.

We define $\varphi \vee \psi$ to mean $\neg(\neg\varphi \wedge \neg\psi)$; that is, φ or ψ, and $\varphi \rightarrow \psi$ to mean $\neg\varphi \vee \psi$; that is, φ implies ψ.

Now we must introduce our modal operators B, D, and G; these will represent belief, desire, and goals. If φ is a wff in our language, then so are

- $\langle t, B\varphi \rangle$ (where t is a constant or variable denoting a time point), meaning that at time t, φ is believed
- $\langle t, D\varphi \rangle$ (where t is a constant or variable denoting a time point), meaning that at time t, φ is desired
- $\langle t, G\varphi \rangle$ (where t is a constant or variable denoting a time point), meaning that at time t, φ is a goal

Now we need to describe some of the characteristics we expect these notions to have; for example, if we belive φ is true (at time t) and believe that $\varphi \rightarrow \psi$ is true (at the same time t) then we believe that ψ is true at time t as well. This keeps our set of beliefs internally consistent. Some properties we want to have:

B2 $\forall t\, (\langle t, B\varphi \rangle \wedge \langle t, B(\varphi \rightarrow \psi) \rangle \rightarrow \langle t, B\psi \rangle)$.

B3 $\forall t_1 \forall t_2 \neg\langle t_1, B\langle t_2, \mathit{false} \rangle \rangle$.

B4 $\forall t\, (\langle t, B\varphi \rangle \rightarrow \langle t, B\langle t, B\varphi \rangle \rangle)$.

B5 $\forall t\, (\langle t, \neg B\varphi \rangle \rightarrow \langle t, B\langle t, \neg B\varphi \rangle \rangle)$.

D1 $\forall t_1 \forall t_2 \neg\langle t_1, D\langle t_2, \mathit{false} \rangle \rangle$.

G1 $\forall t\, (\langle t, G\varphi \rangle \wedge \langle t, G(\varphi \rightarrow \psi) \rangle \rightarrow \langle t, G\psi \rangle)$.

G2 $\forall t\, (\langle t, G\varphi \rangle \wedge \langle t, G\psi \rangle \rightarrow \langle t, G(\varphi \wedge \psi) \rangle)$.

G3 $\forall t_1 \forall t_2 \neg\langle t_1, G\langle t_2, \mathit{false} \rangle \rangle$.

So our agents' beliefs are internally consistent and don't include falsity; agents are aware of their own beliefs (according to **B4** and **B5**; agents don't desire falsity or have it for a goal; an agent's set of goals is closed under implication and conjunction.

How do these notions interact? Whenever an agent has a goal, surely the agent also believes that it has that goal:

$$\forall t\,(\langle t, G\varphi\rangle \equiv \langle t, B\langle t, G\varphi\rangle\rangle)$$

. Intuitively, the agent must have consciously committed to act so as to bring about φ; otherwise φ would only be a desire. (****) Similarly, if an agent thinks that φ is impossible to achieve, then the agent won't choose φ as a goal, because that would mean wasting time trying to achieve something that's impossible:

$$\forall t\,\neg(\langle t, G\varphi\rangle \wedge \langle t, B\neg\varphi\rangle)$$

.

When does an agent form a goal? Presumably, only when achieving that goal will help the agent satisfy some desire the agent already has. So every goal either is also a desire, or will serve to help achieve that desire:

$$\forall t\,(\langle t, G\varphi\rangle \rightarrow \exists\psi\,\langle t, D\psi \wedge B(\varphi \rightarrow \psi)\rangle)$$

. This provides a necessary condition for having a goal; we might also specify a sufficient condition. For example, we might say that any time an agent has a desire and doesn't believe that desire is impossible to achieve, the agent must adopt that desire as a goal.

$$\forall t\,((\langle t, D\varphi\rangle \wedge \langle t, \neg B\neg\varphi\rangle) \rightarrow (\langle t, G\varphi\rangle \vee \langle t, G\neg\varphi\rangle))$$

.

VII. Research on Intelligent Agents

Barbara Hayes-Roth
Stanford University

Paper Prepared for AFOSR Workshop on
Intelligent Real-Time Problem-Solving Systems

Santa Cruz, Ca.

October, 1989

1. Real-Time Performance in Intelligent Agents

Imagine an "errand robot" driving an automobile on its way to some destination. Noticing a yellow traffic light at the next intersection in its path, the robot infers from its current speed, distance to the light, and conservative traffic-light policy that it should stop. The robot immediately releases the accelerator and, after a few seconds, applies the brake to bring its vehicle to a gradual stop just before entering the intersection. The robot's behavior is satisfactory not simply because it produces the correct result, but because it does so at the right time. If, for example, the robot stopped very much before or after reaching the intersection, its behavior would be unsatisfactory and potentially catastrophic.

The errand robot illustrates a class of computer systems, which we call "intelligent agents," whose tasks require both knowledge-based reasoning and interaction with dynamic entities in the environment--such as human beings, physical processes, other computer systems, or complex configurations of such entities. Tasks requiring an intelligent agent occur in diverse domains, such as: power plant monitoring (Touchton88), sonar signal interpretation (Nii82), process control (Allard87, d'Ambrosio87, Fehling86, LeClair87, Moore84, Pardee87,89), experiment monitoring (O'Neill89), student tutoring (Murray89), aircraft pilot advising, and intensive care patient monitoring (Fagan80, Hayes-Roth89a).

To perform such tasks, an agent must possess capabilities for: *perception*--acquiring and interpreting sensed data to obtain knowledge of external entities; *cognition*--knowledge-based reasoning to assess situations, solve problems, and determine actions; and *action*--actuating effectors to execute intended actions and influence external entities. In the example above, the errand robot perceives signals from which it infers that the traffic light is yellow. It reasons with this perception, its traffic light policies, and other perceptions and knowledge to determine that gradually coming to a stop at the intersection is the appropriate result and that releasing the accelerator and applying the brake are the appropriate actions. It performs those actions in the appropriate temporal organization, thereby achieving the intended result.

Because external entities have their own temporal dynamics, interacting with them imposes aperiodic hard and soft real-time constraints on the agent's behavior. Following Baker89, we use the term "aperiodic" to describe tasks having irregular arrival times. Following Faulk88 and Stankovic88b, we use the terms "hard" and "soft" to

distinguish between real-time constraints whose violation precludes a successful result versus those whose violation merely degrades the utility of the result. For example, a vehicle that happens to stop in front of the errand robot is an aperiodic event with a hard deadline. The robot must stop its own vehicle in time to avoid colliding with the other vehicle. When that is not possible, the robot should consider alternative actions, such as maneuvering around the stopped vehicle.

In a complex environment, an agent's opportunities for perception, action, and cognition typically exceed its computational resources. For example, in the scenario above, the errand robot has opportunities to perceive the physical features and occupants of other automobiles on the road and the buildings and landscape along the sides of the road. It might reason about any of these perceptions or other facts in its knowledge base. It might perform a variety of actions more or less related to driving its automobile. Fortunately, the robot largely ignores most of these opportunities to focus on matters related to the traffic light. Otherwise, it might fail to perform the necessary perception, reasoning, and actions in time to stop its automobile at the right time. On the other hand, the errand robot cannot totally ignore incidental information without risking the consequences of rare catastrophic events. For example, the robot should notice a child running into its path. In some cases, the robot might benefit from noticing information that is not immediately useful. For example, it might notice a sign posting business hours on a shop window and use that information when planning a subsequent day's errands.

Because an intelligent agent is almost always in a state of perceptual, cognitive, and action overload, it generally cannot perform all potential operations in a timely fashion. While faster hardware or software optimization may solve this problem for selected application systems, they will not solve the general problem of limited resources or obviate its concomitant resource-allocation task (Stankovic88a). For an agent of any speed, we can define tasks whose computational requirements exceed its resources. Moreover, we seek more from an intelligent agent than satisfactory performance of a predetermined task for which it has been optimized. Rather, we seek adaptivity of the agent to produce satisfactory performance of a range of tasks varying in required functionality and available knowledge as well as real-time constraints. For example, the errand robot should be able to respond appropriately to traffic signals and other usual and unusual events in a broad range of driving situations. It should drive competently on freeways as well as on surface streets. If it unexpectedly finds itself on surface streets where others are driving at freeway speeds (or, more likely, vice versa), it

should adapt its own behavior accordingly. The agent might have other sorts of skills, such as planning its own errands under high-level goals and constraints or learning new routes from experience taking necessary detours. Other things being equal, the broader the range of tasks an agent can handle and the wider the range of circumstances to which it can adapt, the more intelligent it is.

For these reasons, we view real-time performance as a problem in intelligent control. An agent must use knowledge of its goals, constraints, resources, and environment to determine which of its many potential operations to perform at each point in time. For example, the errand robot might decide to give high priority to perceiving and reasoning about traffic lights so that it can always stop in time for yellow or red lights. When the operations required to achieve an agent's current goals under its specified constraints exceed its computational resources, it may have to modify them as well. For example, if the errand robot finds itself unexpectedly late to an important destination, it might decide to relax its conservative traffic-light policy and drive through selected yellow lights. Because it is situated in a dynamic environment and faces a continuing stream of events, an agent must make a continuing series of control decisions so as to meet demands and exploit opportunities for action as they occur. For example, if the errand robot is making a planned gradual stop at a traffic light and a child runs into its path, the robot should perceive the child and stop immediately. In general, an agent should use intelligent control to produce the best results it can under real-time constraints and other resource (e.g., information, knowledge) constraints.

Our conception of real-time performance in intelligent agents is qualitatively different from conceptions of real-time performance in other sorts of computer systems (Baker89, Brinkley89, Faulk88, Henn89, Lauber89, Marsh86). In particular, we do not view real-time performance as a guaranteed, universal, or provable property of the agent. Nor do we seek real-time performance through effective engineering of the agent. We feel that these constructs are surely premature and possibly unrealistic for the versatile and highly adaptive agents we envision. Rather, we view real-time performance as one of an agent's several objectives, which it will achieve to a greater or lesser degree as the result of interactions between the environment it encounters, the resources available to it, and the decisions it makes. In many cases, the agent will achieve real-time performance only at the expense of quality of result or by compromising response quality or real-time constraints on other tasks. Ironically, as the agent's competence expands, so will its need to make such compromises. From this perspective, real-time performance in intelligent agents

depends critically upon an underlying architecture that enables agents to make and apply the necessary kinds of control decisions.

2. Real-Time Requirements and Heuristics

An intelligent agent's real-time requirements can be summarized very simply: To maximize the number of important goals for which it achieves an acceptable result at an acceptable time. This section presents heuristics for meeting these requirements. We do not mean to suggest that these heuristics provide the optimal approach to meeting the requirements or even a valid approach. We suggest only that they represent a promising approach, which we currently are investigating.

1. *Asynchrony.* For a given objective, an agent can't count on all necessary perceptual information being available at the start of its associated reasoning or on its reasoning being completed prior to execution of its first associated action. Relevant information may arrive at any time during reasoning and relevant reasoning may continue beyond initiation of early actions. In addition, with multiple objectives, the agent may have to interleave unrelated perception, reasoning, and action operations. Finally, the agent must always be prepared to interrupt its ongoing acitivites to handle unpredictable emergencies or simply to switch its attention to more important matters than those currently under consideration. For these reasons, an intelligent agent must perceive, reason, and act asynchronously, without these activities interrupting or directly interfering with one another or with communications among them. For example, in the scenario above, the errand robot's execution of actions planned to produce a gradual stop do not impede its immediate perception of a child running unexpectedly into its path or its consequent actions to avoid hitting the child.

2. *Timeliness.* Because an agent interacts with independent dynamic entities, its sensory information is perishable, the utility of its reasoning operations degrades with time, and the efficacy of its actions depends upon synchronization with fleeting external events. Therefore, the agent must perceive present or recent sensed events, perform present or recent reasoning operations, and execute currently intended actions, regardless of how many earlier unexploited opportunities have occurred. The agent should not fall behind real time to handle a backlog of inputs in any of these categories and it should not operate on seriously out of date inputs--unless it has explicitly decided to do so. For example, while approaching an intersection, the errand robot perceives information relevant to its reasoning about actions to be taken at the intersection. Having passed

through the intersection, the robot ordinarily does not dwell upon unprocessed perceptual events available at the intersection, inferences it might have drawn, or alternative actions it might have taken. Instead, it performs operations related to its current post-intersection situation.

3. *Selectivity.* Except for rare occasions, opportunities for perception, cognition, and action vastly exceed the agent's resources for performing those operations. Even when the rate of such opportunities is well within the agent's capacity, it may be unproductive or even harmful for the agent to pursue many of them. Therefore, the agent must selectively perceive information and perform reasoning operations that enable it to perform the most useful actions. For example, although the errand robot could respond to great quantities of incidental information regarding the other vehicles on the road, their drivers' intentions, or the passing landscape, it ignores most of these opportunities in favor of activities related to its current driving task.

4. *Coherence.* Most non-trivial tasks require coordinated perception, cognition, and action. In the simplest case, perceived information must be integrated with knowledge and reasoning to determine actions that lead to objectives. In more complex, but no less typical cases, achievement of objectives depends upon a strategic sequence of such activities coordinated over a period of time. To perform such tasks, the agent must develop strategic plans and use them in a "top-down" fashion to direct its perception, cognition, and action toward the desired objectives. For example, given its general plan for travelling about the city and dispatching errands, the errand robot focuses on dynamically refining its intended route and obeying traffic laws as it follows that route.

5. *Flexibility.* A dynamic environment entails considerable uncertainty in the situations an agent will face and the details of those situations as they unfold. In addition to rare catastrophic events, many unanticipated events simply require minor modifications of the agent's behavior or even offer new opportunities for the agent to improve the overall utility of its performance. In order to adapt to actual evolving situations, the agent must monitor the environment (and its own inferences as well) for important unanticipated events and respond flexibly to those events in a "bottom-up" fashion to modify its objectives or its strategic plans for achieving them. For example, to avoid traffic delays caused by an accident, the errand robot should modify its planned route and take a reasonable detour. If the robot calculates that the detour also entails excessive delay, it might eliminate certain errands from its plan in order to insure completion of critical errands in the time available.

6. *Responsivity*. Situations vary in urgency--that is, the rapidity with which a response is required in order to produce a satisfactory result. Some situations impose "hard" deadlines; a logically correct response that occurs after the deadline is as useless as a logically incorrect response. Other situations impose "soft" deadlines; the utility of a logically correct response declines monotonically, but not precipitously, with delay after deadline. Depending upon the complexity of its environment, the agent may incur multiple aperiodic tasks imposing various hard and soft deadlines within a local time interval. Often it must compromise the correctness or timeliness of its performance of some of these tasks in order to satisfy the requirements of other more important tasks. However in general, other things being equal, the more urgent a situation is, the more quickly the agent should perceive the relevant information, perform the necessary reasoning, and execute the appropriate actions. Thus, the errand robot should perceive a child in its path and stop its vehicle as quickly as possible, while a somewhat longer latency is tolerable for perceiving and stopping at a yellow traffic light.

7. *Robustness*. Despite its best efforts to use its limited resources wisely, an agent will encounter situations that strain or exceed its capacity: too many important perceptions, reasoning tasks, and actions. Regardless of the degree of overload, the agent must continue to function. It cannot arbitrarily ignore pending tasks or fail to complete tasks it has decided to undertake. Instead, the agent must adapt to high resource-stressing situations by ensuring graceful degradation of its performance. It must reduce the demands on its resources by eliminating or revising the least important of its objectives and refocusing its resources where they are needed most. It must compromise the quality of its performance to provide satisfactory results for the most important objectives. For example, when driving near a school or playgound, the errand robot may compromise its responsivity to traffic lights so that it can monitor more vigilantly for children running into its path.

3. Research Goals

The primary goal of our research is to develop a general architecture for *intelligent agents*--systems that perform a variety of knowledge-based reasoning tasks while functioning autonomously in naturalistic environments. By definition, an intelligent agent must integrate capabilities for performing the following tasks: interpretation of information sensed from a complex, dynamic environment; detection and diagnosis of exceptional conditions; reactive response to urgent

conditions; prediction of important future conditions; planning and execution of a course of actions to influence external conditions; explanation of the physical phenomena underlying observed or predicted or planned conditions; consultation with human beings or other computer systems; and learning to improve performance based on experience. In performing these several tasks, the agent must allocate limited computational and other resources (e.g., sensors and effectors) dynamically, to achieve its most important objectives in a timely fashion. In particular, they must achieve at least the real-time performance objectives characterized above.

4. Architecture

In previous work, we developed a blackboard architecture that integrates task-level reasoning with reflective processes, such as dynamic planning and control of reasoning, strategic explanation, and strategy learning. Implemented in the *BB1* system, this architecture has been used in a number of projects at Stanford and licensed widely outside of Stanford. Our current work is directed toward extending the architecture in four areas: perception/action processes to mediate asynchronous interactions with a complex environment, a satisficing control regime to support guaranteed response times, and conceptual graph representation to provide a declarative representation for all of an IRTPS's knowledge, beliefs, intentions, etc., with a temporally organized representation of dynamic information. The architecture is designed to address all of the requirements discussed above. Current research activities are intended to evaluate its achievement of those design goals both analytically and empirically. The architectural design and implementation and related research are discussed in several articles in the attached references, especially Hayes-Roth85, 87b, 88a, 89c.

We are developing generic models of knowledge and reasoning for prototypical tasks performed by an intelligent IRTPS. Examples are: focus of attention, incremental modeling of dynamic external phenomena; reactive detection, diagnosis, and correction of exceptional external conditions; model-based diagnosis, prediction, and explanation of external conditions; time-constrained planning of longer-term courses of action. Our approach represents explicit knowledge of the operations and strategies involved in each of these tasks within the architecture discussed above. Some of this work is discussed in (Boureau89, Hewett89b, Hayes-Roth89b, Washington89).

Much of our past work concerns coordination of diverse knowledge sources and reasoning methods invoked by run-time conditions (Garvey87, Hayes-Roth85, Johnson87) to achieve efficient and effective performance of complex tasks. Current research expands these concerns to performance of multiple interacting tasks under real-time and other resource constraints. In particular, we have developed an approach to integrating reactive response to urgent events in the context of more deliberate situation assessment and planning for stable or slowly evolving conditions.

5. Experimental Research in Testbed Applications

Because our long-term research goal is to develop a general architecture for adaptive intelligent systems, much of our research involves experimental development of application systems in various several domains. Each new domain tests the efficacy and generality of the current architecture and presents new requirements to be addressed in subsequent versions of the architecture. We currently are working on several intelligent real-time monitoring applications, described briefly below.

Guardian, being developed in collaboration with Dr. Adam Seiver, of the Palo Alto Veterans Administration Medical Center, monitors ventilator-supported patients and consults with physicians and nurses in a surgical intensive care unit. Demonstration 4, which will be completed this Fall, monitors about twenty automatically sensed variables (e.g., pressures, temperature) and a few irregularly sensed variables (e.g., lab results). It performs the several tasks mentioned above using several kinds of knowledge: heuristic knowledge related to common respiratory problems; structure/function knowledge of the respiratory, circulatory, metabolic, and mechanical ventilator systems; and structure/function knowledge of generic flow, diffusion, and metabolic systems. Guardian currently monitors a simulated patient-ventilator-hospital system. However, beginning this winter, we expect to have on-line access to patient data monitors at the Palo Alto VAMC, so that we can begin experiments involving real-time monitoring of actual patients. A paper describing Guardian is in preparation.

A second system, being developed in collaboration with Professor James Harris of Stanford's Electrical Engineering Department, is intended to control the growth of gallium arsenide (GaAs) crystals by molecular beam epitaxy (MBE). A simulation of the MBE machine and crystal growth

process have been developed and a preliminary process planner is near completion.

We currently are exploring two other domains: intelligent plant monitoring for preventive maintenance, in collaboration with Professors Raymond Levitt and Paul Teicholz of Stanford's Civil Engineering Department; and intelligent control of micro-factory operations for semi-conductor fabrication, in collaboration with Professor Nils Nilsson of Stanford's Computer Science Department and with Professors Krishna Saraswat, Gene Franklin, and Robert Dutton of Electrical Engineering.

6. Related Work

In this section, we briefly review other research related to adaptive intelligent systems. This is by no means a complete survey, but gives an indication of some of the most prominent lines of relevant research and their treatment of the requirements discussed above.

A number of researchers have extended the classical planning model (Fikes71, Sacerdoti75) to permit interleaving of planning and execution, either to build a plan incrementally or to modify the plan in response to unanticipated conditions (Broverman87, Corkill82, Durfee86, Georgeff87, Hayes-Roth85, Lesser88) or to employ more knowledge intensive and computationally tractable methods for generating partial plans, including for example: instantiating goal-oriented action schemas (Friedland79); integrating top-down and bottom-up planning methods (Hayes-Roth79a,b, Johnson87), transferring prior successful plans to new situations (Hammond86); or successively applying constraints among potential actions (Stefik81)). These approaches are quite compatible with our work.

Several researchers have studied methods for controlling trade-offs in the amount of time spent solving a problem or performing a task versus the quality of the result, (Dean88, Horvitz87, Lesser88). These approaches, which Dean calls "anytime algorithms" and Lesser calls "approximate reasoning," are quite compatible with and appear in several aspects of the proposed architecture, notably in its satisficing control regime and in its accommodation of alternative reasoning strategies, including approximate or anytime strategies.

By contrast, in an effort to avoid the computational cost of control reasoning and thereby create real-time responsivity, a number of researchers have turned their attention to the theory, design, and implementation of "reactive agents" (Agre87, Andersson88, Brooks85,

Fagan84, Firby87, Kaelbling88, Nilsson89, Rosenschein86, Schoppers87, Suchman87). Basically, reactive agents store large numbers of perception-action rules in a computationally efficient form and execute actions invoked by environmental conditions on each iteration of a perceive-act cycle. Thus, they are formally similar to control theoretic methods (Bollinger88, Hale73), where traversal of symbolic networks replaces computation of numerical models. We consider reactivity an essential behavior in some circumstances and our architecture supports it, but we do not consider it a promising foundational architectural principle, first because enumerating all possible perception-action contingencies and encoding them in a computationally tractable form is infeasible for challenging task domains and second because many task domains intrinsically require maintenance of internal state.

For several decades now, robotics researchers have aimed to build "task-level" robot systems (Cox89, Ernst61, Kathib86, Lozano-Perez89). Unlike robots programmed to perform specific mechanical tasks, task-level robots are intended to accept high-level goals and then determine and perform whatever behaviors are necessary to achieve the goals. They are intended to operate under a variety of incidental contextual conditions, including low-frequency exceptional conditions related to hardward, software, or environmental state. Significant applications of this work include efforts to build autonomous vehicles (Burks89, Crowley89, Goto89, McTamaney89). The robotics work is similar in spirit to the present approach--integrating perception, action, and cognition to achieve goals in a real-time task environment--differing primarily in emphasis on perceptual-motor rather than cognitive functions. The robotics research traditionally has focused on challenging perceptual-motor tasks, but has begun to incorporate more cognitive activities, such as goal determination, planning, exception handling, and learning (Bares89, Barhen89, Rao89, Weisbin89). Conversely, our work grows out of earlier work emphasizing reasoning and problem solving, with new emphases on perceiving and acting in a real-time environment.

Finally, there has been interesting theoretical work aimed at a formal characterization of agents (e.g., Genesereth89). This work tends to encompass an extremely broad spectrum of computational entities. However, recent work by Shoham (described in his paper for this workshop) focuses on agents possessing formal versions of knowledge, beliefs, desires, goals, etc., bringing it closer in spirit to the kinds of intelligent agents discussed in this paper

VIII. IRTPS Workshop Interim Team Report

Stanley J. Rosenschein (Teleos Research)
Michael Fehling (Stanford University)
Matt Ginsberg (Stanford University)
Eric Horvitz (Stanford University)
Bruce D'Ambrosio (Oregon State University)

October 30, 1989

1 Introduction

Over the last decade and a half, advances in knowledge-based systems technology have led to practical applications in a variety of problem domains. Despite these advances, current technology remains inadequate for dealing with systems that must maintain real-time interactions with ongoing processes in their environment. Because systems of this type are critical to many important applications, particularly in defense, the Air Force has launched an initiative aimed at stimulating the development of a national research effort on Intelligent Real-Time Problems Solving (IRTPS).

The goal of the first phase of the research initiative is to clarify terms and issues underlying intelligent real-time problem solving, and as part of this effort three research groups, including the Teleos group led by S. Rosenschein, have been chosen to investigate these issues to help guide subsequent phases of the program. Phase 1 is to culminate in a workshop at which the groups

*This work was supported by AFOSR Contract F49620-89-C-0117.

compare their findings and discuss issues with invited researchers. This document contains an interim report of the Teleos group and is intended to serve as a draft position paper for the IRTPS Workshop. It contains a preliminary review of terms and issues followed by a discussion of implications for the IRTPS research program being undertaken by the Air Force.

2 Background

In the simplest terms, intelligent real-time problem-solving (IRTPS) systems are characterized by the following features:

1. They take in a stream of sensory input from the environment.
2. They produce a stream of control output that affects the environment.
3. The intervening computation is modeled as a reasoning or problem-solving process.
4. Time matters.

The primary challenge in developing systems of this type lies in reconciling the conflict inherent in the last two attributes. In conventional knowledge-based applications, the system is intended to provide support to human problems solvers, where the humans have sole responsibility for real-time interaction with the environment, and the system is not required to exhibit real-time performance. For example, these systems are typically unable to adapt their mode of operation to changing deadlines for an action or the unpredicted occurrence of critical events.

The central programming model for conventional knowledge-based systems work is that of an inference engine that performs reasoning steps and draws conclusions from a set of domain-specific rules or facts stored in a knowledge base. Because chains of inference leading to conclusions can vary greatly in length and draw on facts in the knowledge base in ways that are hard to predict and control, it is difficult to bound the execution time of programs in this model. This is compounded by the difficulty of controlling the performance of the operating systems environment (e.g., list-processing) in which most knowledge-based systems are embedded. In fact, much of the appeal of the knowledge-based model lies precisely in the fact that it abstracts

away from the details of resource allocation required to support inference and allows the programmer to deal primarily with the content of the inferences. However, in real-time applications, the resource question cannot be ignored. Without considering execution time, rates of inference cannot be related to rates of change in the environment, and the designer cannot be sure that the system will be able to find satisfactory output responses in a timely fashion.

3 General programmatic issues

We see the IRTPS program as addressing the need for real-time knowledge-based systems by developing three mutually-supportive research thrusts:

1. Historical/Interdisciplinary
2. Core Research on Resource-Bounded Reasoning
3. Experimental Validation

The allocation of funds among and within these thrusts should be at least partially "bottom-up," i.e., it should be responsive to the best ideas offered during proposal solicitation. However, an effort should be made to maintain balance so that major areas are not entirely neglected.

Because of the limited funding allocated to the main research effort of the IRTPS program ($800,000 over an 18-month period), a realistic objective for the program is to seed research in each of the key areas and to establish research paradigms and activities to which additional funding can be attracted. This is especially true for the experimental component of the program; the resources required to develop and distribute a realistic testbed, for instance, could overwhelm the program's funding, leaving little for basic research unless special care is taken to leverage existing software and other government programs.

In the following sections, we describe each of the proposed program thrusts and list several research questions that might be explored in each. These questions are meant to be suggestive rather than exhaustive, and will undoubtedly be supplemented as the research proceeds.

4 Historical/Interdisciplinary

The current IRTPS program does not exist in a vacuum. IRTPS systems have been built using existing artificial-intelligence (AI) concepts and tools, and these should be investigated with a view toward drawing out the lessons to be learned. Among the approaches that might be taken to these investigations are case studies, literature searches, and attempts to classify existing systems in terms of the conceptual categories developed in the IRTPS program.

In addition, artificial intelligence is not the only discipline to be concerned with embedded real-time computation or intelligent problem solving. Work in decision theory and control theory, in real-time operating systems and scheduling algorithms, and in computer-based control systems has resulted in a large body of practice, theory, and engineering methodology. Research in these disciplines should be explored in light of IRTPS requirements and objectives. In particular, methods should be developed that will allow non-AI approaches to be adapted and applied to IRTPS problems and to coexist with specialized techniques developed in the AI framework.

5 Core Research on Resource-Bounded Reasoning

This thrust should be aimed at developing a deeper understanding of the tradeoffs inherent in reasoning under time stress. Constraints on a real-time reasoning system's inference and representation lead to inescapable uncertainties about the problems that may be faced. A real-time system immersed in a complex world must grapple with uncertainty associated with both the environment (object-level) and the reasoner (inference level). In addition, agents must typically contend with deep uncertainty about the value of future reasoning. However the benefits of controlling computational tradeoffs in theoretically coherent ways are very great in high-stakes decision-making arenas such as medicine, aerospace, and defense, and justify an intensive research effort in this area. An important part of this research will involve a synthesis of logical models of reasoning with Bayesian and, more generally, decision-theoretic models.

5.1 Base-level/Meta-level Reasoning

One set of research issues focuses on the control of search as a method for bounding execution time of reasoning processes. Reasoning is modeled as a search process in which many potential branches are available for exploration and in which choices are made about where effort should be allocated. The computation is broken into elementary bounded-time steps (with different approaches varying in the grain size they consider for these elementary units), and attention is shifted under the control of a higher-level process whose time behavior is well understood. This method of resource allocation similar to that used in a multi-tasking operating systems, and, as in the case of operating systems, interruptability, pre-emption, and prioritization are the key terms of analysis. Unlike operating systems, however, issues of the *content* of the computation as a reasoning process need to be more fully modeled and related to the resource-allocation algorithm.

Considerations of resource allocation lead directly to questions regarding the *criteria* by which allocation is to be judged. One approach (metalevel control of reasoning) is to formulate explicit theories about the reasoning process and the effect of alternative control strategies. Research is needed on methods for allocating resources between base-level and meta-level reasoners. An important goal of this research is to discover ways of limiting the amount of time spent doing meta-level reasoning, or more generally, to optimize the split of resources between base- and meta-level reasoning. An attempt should be made to identify classes of tractable, closed-form, metalevel control problems.

For many applications it wil be important to quantify (1) the value or cost of achieving (or not achieving) goals, (2) uncertainty about the existence of alternative states of the world, and (3) the costs of continuing to deliberate (versus taking an action). Decision theory provides a useful framework for the design and evaluation of real-time systems as it gives us a language and precise semantics for capturing preferences and uncertainty.

5.2 Anytime reasoning

Algorithms that can produce partial or reduced-quality output when their execution is terminated before some complete soluction is produced have been called *anytime algorithms.* Useful properties that such algorithms can

exhibit include monoticity (improvement over time), continuity (gradual improvement), and convergence in the limit. We see a need to extend this line of research to include inference processes, so that so that declarative computations can be interrupted before they have finished running and still produce useful results.

Among the approaches that can be tried are the following:

1. Come up with an anytime inference algorithm that gradually and uniformly approaches the right answer.

2. Come up with an anytime inference algorithm that approaches the right answer in the large-runtime limit, but might wander around before doing so. (Presumably, the "quality" of the answer would increase uniformly, in some strange sense.)

If the second approach becomes necessary, one important question will be how to adapt the reasoning process, and the strategies that guide it, to changes in the environment that invalidate previous input or modify the available problem solving resources, Truth-maintenance techniques will be important here, though research will be required to adapt these techniques to the real-time setting. A related set of research topics involve modeling anytime probabilistic and decision-theoretic reasoning.

5.3 Compilation (Compiled vs. Deliberative Reasoning)

So far, we have only discused deliberative approaches to reasoning and metareasoning. It can be important to reduce complex deliberation in computer-based reasoners by developing decision-making techniques that rely to some extent on precomputed or *compiled* responses. Such knowledge can be generated at design time or learned by agents over their lifetimes.

Recent research, including that labeled *reactive planning*, has centered on the replacement of unwieldy solution mechanisms and detailed representations of knowledge with compiled *situation-action* rules. Such rules enable agents to respond immediately perceptual inputs. Investigators have sensed that, for many contexts, explicit representations and deliberation will not be necessary for good performance.

Deliberation and reaction are merely two ends of a spectrum with many intermediate points. One imporant topic of research involves developing methods for optimizing the split between deliberation and reaction in the design of embedded agents.

5.4 Pre-emptive control

This sub-area should explore pre-emptive control strategies for multiple, simultaneous problem-solving activities, with a distinction being drawn between pre-emption and multitask management. For example, pre-emption can occur in managing even a single line of control. The policies that guide pre-emption, and the systems architecture that facilitate the implementation of these policies, need to be characterized and studied.

6 Experimental Validation

While abstract conceptual models of real-time reasoning are an important first step toward the design of practical IRTPS applications, these models must be augmented by a software-development methodology that designers can actually use to solve real-world problems. The methodology should help the programmer instantiate the general model to the particular data structures and operations required to satisfy the requirements of his particular application problem. The methodology includes specialized software-development tools that capture key abstractions, hide implementation details, but leave the programmer with sufficient flexibility and control over what is important. One goal of the IRTPS research program should be to enumerate and taxonomize programming methodologies, architectures, and tools, relating them to the underlying conceptual model they support, and exploring whether the abstractions they provide can be made orthogonal and be incorporated into a more embracing IRTPS software methodology.

One of the goals of the IRTPS program is to provide a methodology for exploring architectures for real-time problem solving in an empirical setting. One way of promoting this goal is by establishing a common conceptual framework to guide experimental work in the IRTPS community. Program management has established an Experimental Methodologies Working Group which has outlined such a framework and produced a draft document de-

scribing its findings. The document describes ways that models of intelligent embedded systems might be modeled and evaluated, focusing specifically on comparing the effectivesness of agents with different compositions and abilities immersed in distinct problem contexts. Evaluation methods include theoretical analysis as well as experimental methods, both in simulated and real environments. The document goes on to describe what kind of controls would be necessary to make the results of experimental methods empirically meaningful and proposes a variety of measurement types relevant to real-time problem solving. Our team has reviewed a preliminary draft of this document and, in general terms, is in agreement with its analysis and recommendations.

A second, more concrete, way of enhancing the field's experimental methodology is to provide a common testbed that might be used by the IRTPS community to carry out empirical investigations. Such a testbed would allow precise measurements of the performance of systems and architectures. We feel such a testbed would be a useful component of the IRTPS program, provided it can be provided at reasonable cost and can be configured to adhere to the methodological guidelines proposed in the Experimental Methodologies Working Group document. One method of leveraging the program's research funds to good advantage would be to identify existing testbeds produced under other government programs that could be adapted to support experimental work in IRTPS. Consideration should be given to dimensions of variability among application domains, for example discrete vs. continuous domains, low-level vs. high-level perceptual data, and so on.

IX. Intelligent Real-Time Problem Solving: Issues and Examples

Paul R. Cohen, Adele E. Howe, David M. Hart
Experimental Knowledge Systems Laboratory
Department of Computer and Information Science
University of Massachusetts, Amherst

1. Introduction

This report presents our work on real time problem solving (IRTPS). The topic is fundamentally challenging in the sense that it probably cannot be completely addressed within the established knowledge-based and logicist paradigms, but will require methodological, theoretical, and technical developments. Accordingly, this report looks at IRTPS from all these perspectives. Because readers will have different interests, each section of the report is independent of all sections except this Introduction. Section 2 offers a definition of IRTPS. Section 3 describes our real-time testbed, including its current status and portability, and our timetable for making it generally available. Section 4 discusses the architecture we have developed for real-time agents and our near-term research goals. Section 5 is devoted to methodological issues, specifically, how characteristics of environments constrain the design of agents (including an assessment of the pros and cons of simulated environments), how to evaluate IRTPS systems, and the need for analytic models of agent architectures.

The task environment for much of our research is a simulation of forest fires. The task is to control simulated fires by deploying simulated agents, including "smart" bulldozers, fuel carriers, and airplanes. (Smart agents have the simulated physical abilities of, say, bulldozers, and some of the simulated mental abilities of their human operators.) This is a realtime problem in the basic sense that *the environment changes while agents think and act.* If agents think too long, the fires get too big to control. If they don't think long enough, their plans may be flawed and their actions may be less effective. (Section 2 refines this basic definition of the real-time problem.)

The Phoenix system comprises five levels of software:

DES -- the discrete event simulator kernel. This handles the low-level scheduling of agent and environment processes. Agent processes include sensors, effectors, reflexes, and a variety of cognitive actions. Environment processes include fire, wind, and weather. The DES provides an illusion of simultaneity for multiple agents and multiple fires.

Map -- this level contains the data structures that represent the current state of the world as perceived by agents, as well as "the world as it really is." Color graphics representations of the world are generated from these data structures.

Basic agent architecture -- a "skeleton" architecture from which agents, such as bulldozers, airplanes, and firebosses are created. The agent

architecture provides for sensors, effectors, reflexes, and a variety of styles of planning.

Phoenix agents -- the agents we have designed (and are designing) for our own RTPS experiments.

Phoenix organization -- currently we have a hierarchical organization of Phoenix agents, in which one fireboss directs (but does not control) multiple agents such as bulldozers. Each Phoenix agent is autonomous and interprets the fireboss's directions in its local context, while the fireboss maintains a global view. A related project is looking at multiple firebosses and distributed control.

The Phoenix environment (the DES and map level), the basic agent architecture, and Phoenix agents are independent software packages that we offer to other researchers (see Section 3.3). We will offer instrumentation for these components of the Phoenix system by the end of Phase I of the IRTPS initiative.

Our research on IRTPS is part of a larger project whose goal is to develop a sound basis for the design of AI agents. We are analyzing agents in terms of the *behavioral ecology view* shown in Figure 1. This view encourages us to ask how the characteristics of environments (including time) constrain the design and behavior of agents. (We compare this view with the S/E model of Rosenschein, Hayes-Roth and Erman, in Section 5). When we speak of a sound basis for design, we mean the ability to predict how modifying the architecture of an agent will change its behavior in a given environment. Currently, we do this by building models that relate the architecture of an agent to behaviors. The methodological implications of the behavioral ecology view and of modelling are discussed in Section 5.

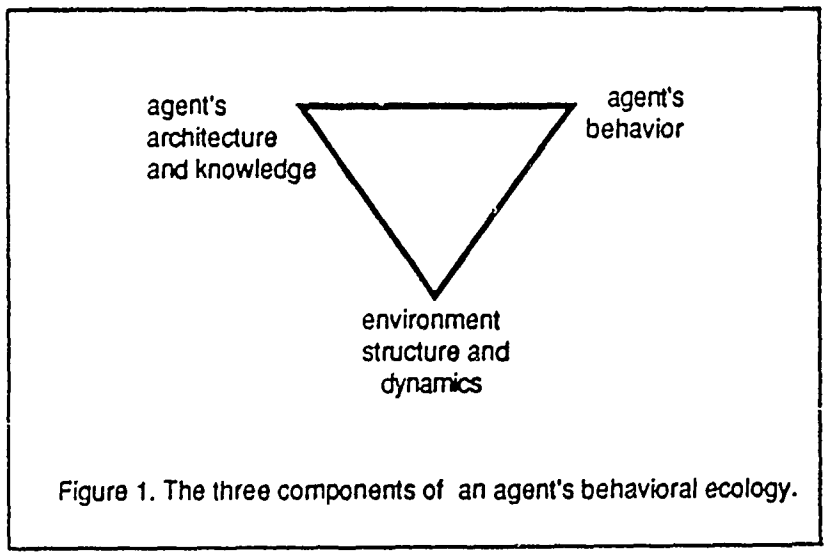


Figure 1. The three components of an agent's behavioral ecology.

2. Definitions.

We begin this section with definitions of real-time problem solving and "the real time problem." Next we examine some terms that are common in the IRTPS literature, such as deadline, predictability, and time scale. These terms are vague, and it is often difficult to tell whether they are intended as descriptions of an agent's environment or its behavior. We propose six classes of terms that should, we hope, reduce the vagueness and ambiguity of previous discussions of IRTPS. Lastly, we show how the behavioral ecology view helps us compare and organize different approaches to IRTPS.

2.1 IRTPS and the Real-time Problem

A crude definition of IRTPS was mentioned in the introduction:

> *The environment changes while agents think and act.*

But this doesn't adequately convey the impact of changes in the environment upon the agents. For example, while a bulldozer thinks about how to avoid a fire, the position of the fire changes, and in some cases the bulldozer can be overrun. A better definition makes explicit the value of problem solving and how it is affected by changes in the environment:

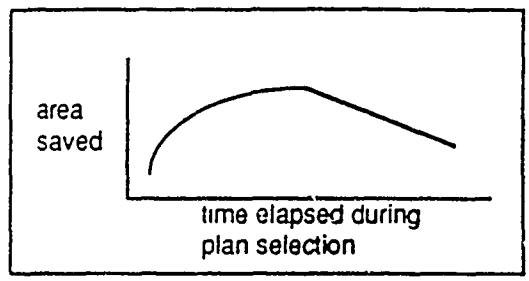
> *The value of problem solving is a function of what the problem solver does --- its thinking and acting --- and one or more parameters in the environment, at least one of which changes during problem solving.*

Note that this definition makes no direct reference to time. This is because time is itself an indirect way of talking about changes in the environment during problem solving, and it is these changes, not the passage of time, that affect the value of problem solving. When we say, "The fire is currently consuming 10 acres per hour," we do *not* mean that an hour is worth ten acres. Time itself has no inherent value. Time provides us a scale on which to measure events that do have value. Consider an analogy to distance. We might say, "As we drive down this street, property values increase by $10,000 a block," but we would not say that a block (e.g., 200 yards) is worth $10,000. Throughout this document we will try to avoid giving the impression that time has any value. We will try to foster the view that time (like distance) is just a scale on which to plot changes in value.

Of course, we can define value to be a function of time, as in real-time operating systems, but typically we measure value in terms of money, acreage, real estate, lives saved or lost, and so on. Unlike time, these value functions may be nonlinear, even discontinuous. This leads to the following definition of the "real time problem":

The value of problem solving does not always increase, nor does it always decrease, during problem solving; thus simple strategies such as "work for as long as possible," or "solve the problem as quickly as possible," will generally not maximize value.

Example: Measuring value in terms of the area of burned forest, we might collect statistics on the relationship between area burned and the amount of time that elapses while the fireboss selects a plan. (For simplicity, we will plot elapsed time against area saved instead of the area burned, so value increases on the y axis:)



Apparently, a little time devoted to plan selection is worthwhile. After a point, however, thinking longer doesn't save more forest. While this "inverted U" could have many causes, the important point is that it seems to be characteristic of real time tasks.

The inverted U function seems more representative of soft deadlines than hard ones. Value decreases slowly when a soft deadline is missed, precipitously when a hard deadline is missed. In Section 2.2 we formalize and illustrate a hard deadline in the Phoenix environment.

It follows from this definition of the real time problem that an agent must have control of the amount of time it devotes to problem solving. In terms of the previous example, an agent should spend "just enough" time on plan selection---the amount of time that corresponds to the highest point on the curve. Although this picture is an oversimplification (we will discuss some of the complexities later) it does illustrate that if an agent cannot control the amount of time it spends on problem solving, it cannot affect the value of its problem solving.

What does it mean for an agent to control the amount of time it spends on problem solving. First, it does *not* mean that the agent controls the rate at which time passes. We assume this is beyond every agent's control. Instead we mean that an agent can with "one eye on the clock" decide whether to run a process or continue running an interrupted process.

Example: The Phoenix agent architecture provides for multiple *execution methods* to achieve any goal. For example, Phoenix has several path planning methods. Execution methods for a given task require different amounts of time. More precisely, they differ in the amounts of time that are expected to elapse before each terminates. One way that Phoenix agents control the amount of time they spend on problem solving is to base the selection of execution methods on their estimated time requirements.

Estimates figure heavily in this example, and in IRTPS in general. What remains to be seen, in the course of this research, is what characteristics of the environment affect the quality of estimates, and what characteristics of IRTPS architectures affect their dependence on the quality of estimates.

2.2 Describing the Environment and Agents

Although our approach is to design agents for specific environments, we have been content to describe environments at two levels of abstraction, both inadequate for design. We believe this is common in the IRTPS literature.

The levels, with examples, are:

Implementation-specific: When a cell ignites, the simulator figures out when its knights-tour-neighbors are going to ignite. It calculates the rate of spread of the newly-ignited cell to its neighbors, accounting for weather, slope, fuel type, etc.

Apple-pie general: The Phoenix environment is characterized by unpredictable events, real-time constraints, and hard and soft deadlines.

Neither level of description of the environment is appropriate for design. From the first kind of description you can model the structure and dynamics of the environment, so it is genuinely useful. But the second kind of description is actually misleading without a lot of clarification, as the following examples show.

At the IRTPS Workshop[2], the Working Group on Architectures made a list of characteristics of environments:

lots of data

low signal to noise ratio

unpredictable rates at which data arrive (varying quantity of data)

hard and soft deadlines

time-dependent value

spectrum of predictability

---

[2]Pasatiempo, Santa Cruz, California. November 6 & 7, 1989.

incompleteness in data

multiple time scales

combinatoric proliferation of things to attend to

Most of these seem self-explanatory. However, most could be interpreted as descriptions of the agent as well as descriptions of the environment. Take "multiple time scales." Our definition of time scale is the average time between causal event cycles that have value for the agent. Some cycles are very short (e.g., the time between moving into a fire and getting burned) and some are much longer (e.g., the time between a wind shift and the recognition of failure of a fire-fighting plan). But it doesn't make sense to talk about time scales independent of an agent; specifically, independent of the value of events to the agent. Without the concept of value, there's no way to classify the limitless number of causal event cycles, and so the distribution of time scales is uniform. The concept of value enables us to select classes of events—those that have value to the agent—and compute the average length of their causal event cycles. Scale depends on the agent design. It is not an inherent property of environments[3].

Which of the other characteristics listed above is inherent to environments, and which depend on the agent design? For some, both interpretations make sense. When we say "lots of data," we could mean two things: First unlike environments like the blocks world, a lot is happening in the Phoenix environment—there's a lot for the agent to attend to. This seems to be a description of an inherent characteristic of the environment. But we might also mean that in this environment, the agent's sensors can take in more stuff than it can process. So "lots of data" can be interpreted as a characteristic of the environment or as a potential problem for the agent.

Next, consider the "predictability" characteristic. When we say events are unpredictable, we are usually mean "unpredictable for this agent." But, again, we might mean "unpredictable for *any* agent." Once again, we must take care to separate the inherent characteristic of the environment—the one that would constrain any agent—from the potential problem that the environment poses the agent.

The resolution of this ambiguity should provide us with the appropriate level of description of environments for doing design.

There is a predicate called, U_e, that takes environmental events as arguments. $U_e(e)$ means that no agent can predict event e.

There is a predicate U_a that takes environmental events and agents as arguments. If $U_a(e,a)$ then agent a cannot predict event e.

---

[3]We are grateful to Les Gasser for pointing this out.

By definition, $U_e(e)$ implies $U_a(e,a)$ for all a. But it is not the case that $U_a(e,a)$ implies $U_e(e)$. This means, at the very least, that we have to be careful when we say an environment is characterized by unpredictability. More importantly, it points to a gap in our understanding of the agent-environment interaction: If $U_a(e,a)$ and not $U_e(e)$, there must be something about event e that makes it unpredictable to agent a, and if we want to design an agent a' for which Ua(e,a') is false, we have to know why Ua(e,a) is true. To designers, it helps to know $U_e(e)$, but knowing $U_a(e,a)$ doesn't help us fix the problem. We need to know something else. For example, if changes in the position of the fire are unpredictable to an agent, and we view this as a problem, then we need to know why the changes are unpredictable. Two contributing factors may be the limited field of view of agents and the statistical distribution of changes in wind speed and direction. One is an architectural characteristic, the other an environmental characteristic, and together they produce U_a(fire-position,agent).

Terms like "unpredictable" are just shorthand for problems faced by particular agents, not characteristics of the environment, except when they are universally quantified over agents. From the standpoint of design, they don't help us much. Instead we will develop a vocabulary to describe environments in *problem-independent* terms. When we say that an event has a statistical distribution we do not imply anything about the architecture of an agent, or anything about the problem that may arise for an agent as a result of the event having a statistical distribution. We suggest six classes of terms:

Environment characteristics (ECs). These are problem-independent, architecture independent descriptors of the environment. For example, a parameter (say, windspeed) changes aperiodically. A counterexample: Windspeed is unpredictable.

Architecture characteristics (ACs). These are problem-independent, environment-independent descriptors of the architecture. For example, the architecture has a random access memory of limitless capacity; or, the plan selection mechanism is bounded in computation time. A counterexample: the error recovery mechanism exhibits graceful degradation. This is a counterexample because graceful degradation implies something about the problem you are trying to solve.

Problems. A problem is a shorthand for an undesirable behavior, that is, an undesirable interaction between a particular agent and a particular environment. For example, unpredictability is a shorthand for interactions in which, because an agent did not anticipate an environmental event, some negative consequence occured.

Inherent problems. An inherent problem is a problem that we believe all agents face, that is, an undesirable interaction between *any* agent and a particular environment.

Solutions. A solution is a shorthand for a desirable behavior that we, as designers, want to see instead of some undesirable behavior—the

problem. For example, a fast sense-act loop is sometimes a solution to the unpredictability problem.

Solution realizations. A solution realization is one or more architecture characteristics or modifications to architecture characteristics; in short, what we intend to do to the architecture to ensure that the solution (which is a behavior, remember) will occur when we want it to.

In Section 5.1, we illustrate how design of a real-time mechanism proceeds from an informal description of a problem, through a formal description in terms of ECs, ACs and Problems, to solutions and solution realizations.

2.3 The Behavioral Ecology Triangle Organizes and Justifies IRTPS Approaches

We can characterize the dozens of approaches to IRTPS in terms of the behavioral ecology triangle in Figure 1. First, what behaviors do we want from IRTPS systems? Second, what characteristics of the environment make particular behaviors necessary or desirable? Third, what architectural decisions can designers make to achieve the desired behaviors in the given environment?

Example: A desired behavior is for Phoenix agents to meet their deadlines. Two characteristics of the Phoenix environment make this desirable: First, most fires must be contained by the coordinated efforts of several agents. Second, fires spread in such a way that if one agent is very late, the work of others is jeopardized. Another characteristic of the environment conspires against coordinated effort: unpredictable changes in parameters such as wind speed and direction differentially affect the progress of agents. The architectural decisions that allow Phoenix agents to meet deadlines despite unexpected events are discussed in Section 4.2.

The behavioral ecology view provides a framework for organizing the real time literature. For example, we can ask what characteristics of the environment make anytime or approximate processing behaviors necessary or desirable, and what architectural choices are needed to implement anytime or approximate behaviors in particular environments. But we have found that the principal advantage of the behavioral ecology view is that it forces us to justify our design decisions in terms of agents' environments.

Example: It is not uncommon to claim that IRTPS requires a behavior called "graceful degradation." (Other candidates are anytime or approximate behavior). Too often, the next step is to build an architecture that implements this behavior in some environment. This is backwards. The first step must always be to ask whether the environment makes graceful degradation (or anytime, or approximate behavior) necessary or desirable.

3. A Real Time Testbed.

This section describes describes the Phoenix testbed from several perspectives. Section 3.1 describes how the testbed appears to a user. Section 3.2 focuses on how the discrete event simulator manages simulation time and cpu time for multiple pseudo-parallel processes. Section 3.3 describes three levels of instrumentation for the testbed. Section 3.4 presents a "baseline scenario," that can be run again and again under different conditions to test real-time architectures. Section 3.5 addresses portability issues. The structure and implementation of the testbed is independent of the architecture of Phoenix agents; indeed, we hope that other researchers will use the testbed as an environment in which to test their own agent architectures. For this reason, we will postpone discussing the Phoenix basic agent architecture until Section 4.

3.1 The Appearance and Behavior of the Testbed.

If you watch the Phoenix system run, this is what you will see: A color representation of Yellowstone National Park, in which fires are spreading and several bulldozers, fuel-carriers, and other agents are travelling and cutting fireline. You will see different kinds of vegetation coded by color. You will also see roads and rivers of different sizes, elevation lines, lakes, houses, and watchtowers. Status windows present elapsed time, wind speed, and wind direction. You have full control over the resolution of your view; for example, you can see the entire map at low resolution or just a few acres at high resolution. You can see the environment as it really is, and as it is perceived by one or more of the agents (which have limited fields of view). Figure 2 shows a view of an area of the park, unfortunately not in color (but see [Cohen, 1989] for color pictures). The grey region at the bottom of the screen is the northern tip of Yellowstone Lake. The thick grey line that ends in the lake is the Yellowstone River. The Grand Loop Road follows the river to the lake, where it splits. The Smokey the Bear symbol in the bottom left corner marks the location of the fireboss, the agent that directs and coordinates all others. Two bulldozers are shown cutting fireline around a fire in this figure. Two other bulldozers are parked near the fireboss, along with a plane and a fuel carrier.

The Map level of the Phoenix environment, from which the graphics representations of the environment are generated, is constructed from Defense Mapping Agency data. Because it includes ground cover, elevation, moisture content, wind speed and direction, and natural boundaries, we have been able to construct a moderately realistic simulation of forest fires (but see Section 5 for a distinction between realism and accuracy). For example, real fires and our simulated fires spread more quickly in brush than in mature forest, are pushed in the direction of the wind and uphill, burn dry fuel more readily, and so on. These conditions also determine the probability that the fire will jump fireline and

natural boundaries; and the intensity of the fire (which is coded by color in the simulation) The physical abilities of fire-fighting agents are also simulated accurately; for example, bulldozers move at a maximum speed of 40 kph in transit (on the back of a truck), 5 kph traveling cross-country, and 0.5 kph when cutting fireline.

Recently, we have implemented some realistic weather factors, specifically, lightning strikes which start fires with some frequency, and rain, which affects the moisture and thus the friability of fuels.

Fires are fought by removing one or more of the things that keep them burning: fuel, heat, and air. Cutting fireline removes fuel. Dropping water and flame retardant removes heat and air, respectively. In major forest fires, controlled backfires are set to burn areas in the path of wildfires and thus deny them fuel.

In the past, fire-fighting agents were inexhaustible, but recently we have started to model their consumption of resources. The following example of monitoring fuel levels and refueling conveys the flavor of problem solving within and among Phoenix agents.

Example: Bulldozers monitor their own fuel levels and notify the fireboss when their tank drops below a preset level. Upon reciept of the "I'm low on fuel" message, the fireboss marks the bulldozer with a status of "needs-refueling" and when the bulldozer becomes idle (i.e. completes the segment of fireline it is working on), the fireboss selects a refueling plan for that bulldozer. This involves allocating an available fuel-carrier, calculating a rendezvous point on a road near the bulldozer, telling the bulldozer where to go and who to look for, telling the fuel-carrier where to go, and waiting for an acknowledgement from the bulldozer that it has received fuel. The bulldozer and fuel-carrier then interact through the following process: the bulldozer notices the rendezvous, requests service, and waits for a service-complete acknowledgement from the fuel-carrier. The fuel-carrier arrives at the destination, waits for service requests, queues them up if necessary (not currently utilized), transfers fuel via a pump-effector, terminating when either the bulldozer is not present or leaves, the refueling tank goes dry, the bulldozer's tank is full, or the requested amount is pumped. The fuel-carrier then tells the bulldozer it has finished and the bulldozer in turn tells the fireboss that the refuleing task has been completed.

3.2 The Implementation of the Testbed

Underlying the Phoenix testbed is a discrete event simulator (DES) that creates the illusion of a continuous world, where natural processes and agents are acting in parallel, on serial hardware (currently a Texas Instruments Explorer II Color Lisp Machine, but see Section 3.5). In the simulation, fires burn continuously over time and agents act in concert to control it. Some of these actions are physical, as

in digging fireline and cutting trees. In parallel to these physical actions, agents perceive, move, react to perceived stimuli, and think about what action(s) to execute next.

The DES manages two types of time: cpu time and simulation time. CPU time refers to the length of time that processes run on a processor. Simulation time refers to the "time of day" in the simulated environment. The illusion of continuous, parallel activity on a serial machine is maintained by segregating each process and agent activity into a separate task and executing them in small, discrete time quanta, ensuring that no task ever gets too far ahead or behind the others. The default setting of the synchronization quantum is five simulation-time minutes, so all tasks are kept synchronized to within five simulated minutes of one another.

The quantum can be increased, which improves the cpu utilization of tasks and makes the simulator run faster, but this increases the simulation-time disparity between tasks, magnifying coordination problems such as communication and knowing the exact state of the world at a particular time. Conversely, decreasing the quantum reduces how "out of synch" processes can be, but increases the running time of the simulation.

Within the predefined time quantum, all simulated parallel processes begin or end at roughly the same simulation time. Types of tasks differ in how they are "charged for" cpu time and simulation time. Sensory tasks run for very short intervals of simulation time, after which they are rescheduled; this gives them a high sampling rate compared to the rate at which the world is changing. Effector tasks may use very little simulation time, or the full synchronization quantum. Fire tasks always run for the full synchronization quantum.

All these tasks are allotted as much cpu time as they need by the DES; there is no constant proportionality between the simulation time and the cpu time they require. To see why, note that fires are implemented as cellular automata, so that the cpu time required to calculate the spread of the fire depends on the size of the fire. It may take only a fraction of a second of cpu time to calculate five simulation-time minutes of burning for a small fire, but several cpu seconds to calculate the five simulation-time minutes for several large fires. Similarly, the amount of cpu time required to calculate a few simulation-time seconds of sensor processing depends on the type of sensor being simulated, so there is no constant proportionality between simulated sensor time and cpu time.

In contrast, there is a constant proportionality between the cpu time allocated to cognitive tasks and simulation time. This is because we want to "charge" agents at a fixed rate for thinking. Because cognition and other processes, such as the fire, are simulated parallel processes, they are always allocated the same amount of

simulation time. So when both have run for their allocated times

elapsed-simulation-time(cognition) = elapsed-simulation-time(fire)

as measured by the simulation time clock. Although these processes could take arbitrary amounts of cpu time, it is advantageous to impose a strict relationship between cpu time and the simulation time of the planner and the fire. Thus,

elapsed-simulation-time(cognition) = k * cpu-time(cognition)

and, from the previous expression,

elapsed-simulation-time(fire) = k * cpu-time(cognition)

The advantage of this proportionality is that we now have a way to exert time pressure on cognition. The *real-time knob* is the device that exerts pressure, simply by increasing k. Clearly, we can change k without changing the amount of cpu time allocated to cognition, and when this happens, the net effect is to increase the amount of simulation time allocated to the fire. Because of the strict proportionality between simulation time and cpu time for cognition, the indirect effect of increasing k is to *reduce the amount of simulation time allocated to cognition, relative to the simulation time allocated to the fire.* That is, to increase time pressure on cognition. Currently k = 300, which means that one second of cpu time for cognition is matched by five minutes of simulation time for the fire. If we increase k to 600, then the fire is allowed to burn for 10 minutes for every cpu second of cognition time.

Cognitive tasks are allotted a full synchronization quantum each time they run. At times there are not enough cognitive activities to fill a quantum, in which case the task ends and waits to be rescheduled. Some cognitive activities take longer than a full quantum, in which case their internal state is saved between quantum steps.

Example: Imagine it is now 12:00:00 in the simulated world, and an agent is about to begin planning. After one cpu second, simulation time for the agent is 12:05:00. The fire is thus "owed" five minutes of simulation time. But before it runs, the DES runs all sensor, effector, and reflex tasks. After that, it may take 7 cpu seconds to calculate the effects of five minutes of fire. Moreover, simulation time is still 12:05:00, because the agent and the fire are simulated parallel processes. So after roughly eight cpu seconds (one for the planner, negligible time for sensors, effectors, and reflexes, and seven for the fire), we have simulated five minutes of planning and five minutes of fire, and both processes are paused at 12:05:00.

3.3 Instrumentation of the Testbed

The Phoenix testbed is designed to support experiments with a variety of IRTPS architectures---not only our Phoenix agent architecture. Currently it has been instrumented to some extent, and much more instrumentation is planned. In this subsection we describe three levels of instrumentation suggested by Nort Fowler. Low level metrics are largely hardware-dependent estimates of how the software system is utilizing the hardware. Middle level metrics give us a fine-grained picture of how a specific architecture behaves over time; for example, we can measure the communication overhead among agents, the time required to respond to significant changes in the environment, the amount of time spent in error recovery, the ability of scheduling algorithms to meet deadlines, and so on.are specific to the agent architecture. High level metrics are domain specific. They record features of the environment that are affected by the agents, such as acreage burned by fires, and consumption of resources.

Any researcher who implements a new agent architecture in the Phoenix environment will have to define middle-level metrics, because these are architecture specific, but probably won't have to define high and low level metrics. For example, Phoenix agents maintain a timeline of pending actions, and we need to know the average latency between posting and executing an action. An agent architecture implemented as a blackboard system may instead look at the scheduling of tasks on an agenda. Most of our middle level metrics are for the Phoenix agent architecture, not for unanticipated other architectures.

Currently, the following instrumentation is complete or nearly so:

Low Level Instrumentation

Run time , Cpu time , Disk wait time, Time since last run , Idle time, Utilization, Overall. Each of these is graphed against time.

We also provide an interface to the Explorer performance metering tools which work at the function call level and provide for each function the: Number of calls , Average run time, Total run time, Real time, Memory allocation, Page faults

Middle Level Instrumentation

Statistics on cpu utilization by the cognitive component of each agent to see the actual profile of real-time response, graphed against time

The latency between when actions on the timeline become available for execution and when they are executed.

Metrics on sensor and effector usage

Metrics on reflexes

The goal of these metrics is to compare the utilization of cognitive and other resources in different scenarios (see Section 3.4). Each scenario will contain important events (e.g., a fire is detected). We will graph these metrics against time and annotate the graphs at the points that the significant events occured, so we can see how the agent architecture responded.

High Level Instrumentation.

Fire destruction is currently measured by amount and type of forest, houses, and agents burned.

Resource allocation is currently measured by amount and type of agents employed to fight the fire, gasoline consumed, fireline cut, distance traveled, and time required to contain the fire.

3.4 Baseline Scenarios

One advantage of studying IRPTS in a simulated environment is the ability to run the same environmental scenario again and again while modifying aspects of the agent architecture (see Section 5). We have recently implemented the ability to define *scripts*, which include the type and number of available agents, and guide the environment through a series of changes in conditions such as windspeed and other weather conditions. We also have the ability to introduce stochastic factors into scripts, such as lightning strikes. Besides scripts, we will soon be able to provide baseline statistics on events such as rates of spread of fires in different conditions.

Scripts play an important role in evaluating IRTPS systems, and comparing IRTPS architectures. By design, scripts can force an agent to confront virtually any IRTPS issue. Here is a simple script that raises the six IRTPS issues that were discussed in the original IRTPS Initiative:

Materiel:

One fireboss to coordinate the activities of three bulldozers

Bulldozers can move 6.5 kph in softwood, 56 kph on road, .5 kph while building line

One watchtower at location approx 42500x37500

Environment:

A fire of radius 700 km starting at coordinates approx 45000x46250,

Starting wind speed and direction 3 kph from the south

Environmental Changes:

At time 2 hr, wind changes to 10 kph from the NW, threatening buildings---the base and lodge

Since it involves a burning fire, the script requires agents to produce relevant output in a timely fashion. The particular scenario includes an environmental

change (asynchronous with the reasoning system) that invalidates previous input, necessitating the detection of a new threat to higher priority areas and a redirection of ongoing reasoning in order to protect them. To handle this scenario, a system must reason efficiently and effectively about temporal processes, namely the expected progress of a fire under particular environmental conditions and the abilities of a limited number of agents to take steps, over time, toward putting out the fire.

We must add that this script confounds the current implementation of Phoenix agents. They are currently incapable of redirecting their efforts to save the base and lodge.

3.5 Portability.

The Phoenix system runs on color and monochrome Texas Instruments Explorers and MicroExplorers. We can package everything together (including support-code for the frame system, grapher, EKSL utilities, etc.) as needed. We are making progress on the documentation.

The entire Phoenix system is designed to be modular, so fellow researchers can use the components they want. The smallest self-contained module, and the most basic, is the Phoenix environment. This includes the DES, the Map layer, and the user interface. We are confident that a researcher could take this code and build his or her own agents to interact with it. However, we have not done this in our own lab, so we cannot be sure.

Above the environment are three additional levels of software---the Phoenix basic agent architecture, our own Phoenix agents, and the organizational structure that holds among our Phoenix agents. The basic agent architecture is a skeleton with hooks for sensors and effectors, reflexes, and a cognitive component (see Section 4). Some weeks ago the entire lab went through the exercise of defining a new type of Phoenix agent (an airplane) given only the basic agent architecture, and we are confident other researchers can do the same. There are really two aspects to defining a new agent. One is mostly bookkeeping: We define frames for the agent that describe its physical abilities, so that the DES knows how it behaves over time. We also define frames that add instances of the new agent to a script. This is the easy part. The hard part is defining the cognitive abilities of the agent. In terms of the basic Phoenix agent architecture, this means defining plans, execution methods, a cognitive scheduler, and other architectural components discussed in Section 5.

Of course, the Phoenix environment does not and should not care about the cognitive component of a new agent, other than to schedule its processes to guarantee the illusion of simultaneity with other agents and environmental

processes. Thus, it is relatively easy to tell the Phoenix environment about the physical abilities of new agents, as in the examples above, and unnecessary to tell the environment how the cognitive components of the agents work. We hope this will make it easy for researchers to use the Phoenix environment, or the environment and the basic agent architecture, to design and test their own agents.

4. Toward a Solution: The Phoenix Agent Architecture

A uniform agent architecture is shared by all agents. This architecture is the structure of the agent, the "hardware" that dictates the fundamental faculties and limitations of the agent. The structure endows and bounds acuity, speed of response, and breadth of action. The structure constrains what an agent can do, but not what it does. Specific methods control what the agent does. Control methods determine what to do and how to do it. This dichotomy between structure and control is reflected in this subsection and the one following it. Section 4.1 describes the agent architecture and Section 4.2 focuses on techniques for real-time problem solving. For a more detailed description of these components, see [Cohen, 1989])

4.1 Phoenix Agent Architecture

The agent architecture has four components. *Sensors* perceive the world. Each agent has a set of sensors, such as fire-location (are any cells within my radius-of-view on fire?) and road-edge (in what direction does the road continue?). *Effectors* perform physical acts such as moving or digging fireline. *Reflexes* are simple stimulus-response actions, triggered when the agent is required to act faster than the time-scale for the cognitive component. An example is the reflex of a bulldozer to stop if it is m' ing into the fire. The *cognitive* component performs mental tasks such as planning, monitoring actions, evaluating perceptions, and communicating with other agents. Although every agent has these components, each component can be endowed with a range of capabilities.

Sensors get input from the world (fire simulation and map structures). Their output goes to state memory in the cognitive component, and also to the reflexive component (triggering instant responses in the form of short programs to the effectors). For example, a bulldozer sensor that detects fire within its radius-of-view updates state memory automatically. If the detected fire is in the path of the bulldozer, the emergency-stop reflex is also triggered. Effectors are programmed by the cognitive component and by reflexes. Their output performs actions in the world. In the preceding example, the emergency-stop reflex would program the movement-effector of the bulldozer to stop. If the fire were not too close, the cognitive component might then step in and program the movement effector to start moving parallel to the fire. If the cognitive component also programmed the blade effector to put the blade in the down position, the bulldozer would not only

maintain a safe distance from the fire, but it would also build fireline as it moved. Sensors and effectors are first-class objects whose interactions with other components and the world are implemented in Lisp code. Reflexes, as mentioned, are triggered by sensory input, which causes them to program effectors to react to the triggering sensation. They are implemented in production-rule fashion, with triggering sensations as their antecedent clauses and effector programs as their consequents. Because they respond directly to the environment and so must keep up with it, sensors, effectors, and reflexes operate at the same time scale as the simulation environment and are synchronized as closely as possible within the discrete event simulator.

The cognitive component receives input from sensors and sends programs to the effectors to interact with the world. It is responsible for data integration, agent coordination, and resource management, in other words, most problem solving activity. This component operates in larger time slices than the others, thus reducing the overhead of context switching, but increasing the possibility of reasoning with outdated information.

The Phoenix cognitive component directs its own actions by adding prospective actions onto the timeline, a structure for reasoning about the computational demands on the agent, then selecting and executing these actions one at a time. Actions may be added in response to a change in environmental conditions (e.g., a new fire) or as part of the computation of other actions (e.g., through plan expansion). Every action that the cognitive component accomplishes is represented on the timeline with its temporal relations to other actions and resource requirements (e.g., processing time and necessary data). The cognitive scheduler decides which action to execute next from the timeline and how much time is available for its execution.

Actions may perform calculations, search for plans to address particular environmental conditions, expand plans into action sequences, assign variable values, process sensory information, initiate communication with other agents, or issue commands to sensors and effectors. These actions are represented in skeletal form in the plan library. Actions are described by what environmental conditions they are appropriate for, what they do, how they do it (the Lisp code for their execution, called the execution methods), and what resources and data, environmental and computational, they require. A plan is a special type of an action. It includes a network of actions related by their data references and temporal constraints.

Planning is accomplished by adding an action to the timeline to search for a plan to address some conditions. When the search action is executed, it selects an action or plan appropriate for the conditions and places it on the timeline. If this new action is a plan, then when it is executed it expands into a plan by putting its sub-actions onto the timeline with their temporal inter-relationships. If it is an action, it instantiates the requisite variables, selects an execution method (there

may be several with differing resource requirements and expected quality of solution), and executes that method. We call this style of planning *skeletal refinement with lazy expansion.* Plans are represented as shells that describe what types of actions should be executed to achieve the plan but do not include the exact action or its variable values until it is executed. Delaying expansion allows the expanded plan to address more closely the actual state of the environment during execution.

This planning style is common to all agents in the Phoenix planner, though it is flexible enough so that agents with a variety of cognitive capabilities are possible. For example, the fireboss has far more sophisticated methods for gathering and integrating information than the bulldozer does. It can direct the actions of the bulldozers, while the bulldozers can only make requests of the fireboss. However, the fireboss, unlike the bulldozers, does know how to get out of the way of the fire because it does not work close to the fire.

Creating a different type of agent requires defining a cognitive component. One can optionally define a set of programmable sensors and effectors (of arbitrary complexity) and add a set of reflexes to handle situations that require instant response by the agent. To create sensors and effectors, the simulator must be told rates of action under varying environmental conditions, range of perceptions, and other physical capabilities. Creating reflexes involves describing the triggers, the expected output from sensors, and the response, the programming for the effectors. The default cognitive component consists of plans, which are networks of actions available to the agent and tailored to situations in the environment, and methods which describe how to execute the actions. Creating a new cognitive component with the same structure as that described here involves defining a new plan library.

Several design decisions in the Phoenix agent architecture have been made specifically to facilitate real-time control. One important decision is to incorporate both reflexive and cognitive abilities in agents, enabling agents to respond reflexively to events that occur quickly, while responding more deliberately to resource management and coordination problems on a longer time scale. The combination of a reflexive and cognitive component accounts for time scale mismatches inherent in an environment that requires micro actions and contemplative processing. Micro actions, such as following a road and keeping out of the immediate range of the fire, involve quick reflexes and little integration of data. Contemplative processing, such as route planning, involves long search times and integration of disparate data such as available roads, terrain conditions, and fire reports. This horizontal decomposition ensures that the agent can perform reflex actions to keep it from danger and maintain the status quo, while also performing more contemplative actions. This strategy for responding to disparate demands of the environment is advocated by Brooks, and Kaelbling; although in both cases, they chose more levels of decomposition for their domains. Our agent architecture, in effect, combines two different planning components.

one highly reactive, triggered by specific environmental stimuli and operating at very small time scale, and the other slower and more contemplative, integrating large amounts of data and concerned with resource management and coordination.

Another design feature that facilitates real-time control is the timeline and its single representation for all actions. Because prospective actions share a uniform representation on the timeline, all problem solving actions have access to the same memory structures and can be monitored and allocated resources using the same mechanisms. All problem solving tasks are subject to the same constraints with respect to resource allocation: how much time is required, what information gathering resources are required, and what data is necessary. This framework allows new cognitive capabilities to be integrated easily by defining their requirements within the action description language and relying on the timeline and its supportive scheduling mechanisms to temporally arbitrate their allocation.

Lazy skeletal expansion also facilitates real-time control. Plans are only partially elaborated before the agent acts. This deferred commitment exploits recent information about the state of the world to guide action selection and instantiation. Completely deferred commitment, such as in reactive planning, is probably not tenable when agents or actions must be coordinated or scarce resources managed. The integration of planning and acting in Phoenix is designed to be responsive to a complex dynamic world by postponing decisions on exactly what action to take, while also grounding potential actions in a framework (skeletal plans coordinated on the timeline) that accounts for data, temporal and resource interactions.

4.2 Real-Time Control in the Agent Architecture

How does a Phoenix agent respond to real-time pressure? One approach is to control processing requirements. This enhances the flexibility of actions and the sophistication of control decisions. Providing alternative execution methods for timeline entries ensures a range of choices that vary in their timeliness. Different scheduling strategies for managing the actions on the timeline provide greater responsiveness to real-time constraints. Another approach is an expectation-based monitoring technique that reduces the overhead of monitoring while providing early warning of plan failure. Earlier warning of plan failure affords the planner more time to adjust and more flexibility in possible responses. These approaches are discussed below.

4.2.1 Control of Processing Requirements

Processing requirements can be controlled in two ways: by controlling how much time is used by individual actions and by controlling the overall distribution of time across all actions. Approximate processing and anytime algorithms are methods

for controlling how much time is used by individual actions. In these methods, processing time is traded against quality or correctness of solution to satisfy time constraints that could not be managed under rigid processing demands. In Phoenix, these methods are alternative execution methods. Execution methods, as introduced in Section 4.1 are lisp code that performs the cognitive actions. Each cognitive action may may be executed by one of several execution methods, with differing time requirements and so differing solution expectations. The Phoenix planner delays the choice of an action's execution method until the cognitive scheduler selects the action for execution, thereby allowing the scheduler to select a method suited to existing time constraints. By postponing the ultimate commitment of cognitive resources until a choice must be made, those resources can be allocated judiciously.

Alternative execution methods are particularly useful in actions that incur potentially high computation costs with predictable results, such as path planning. Phoenix uses an A* algorithm to calculate paths for bulldozers. It searches the two-dimensional map representation of the world for the shortest travel time path between two points. It expands the current best path incrementally, searching each unobstructed neighboring cell for the best next step. The algorithm is parameterized to work at multiple levels of resolution, so that search steps could range from 128 meters up to 8 kilometers. A small search step, 128 meters, yields the shortest path, requiring the least travel time for the bulldozer. However, this resolution requires the most computation (i.e., cognitive resources). The largest search step, 8 kilometers, typically yields a longer path, which requires more travel time, but can be calculated quickly, consuming less computation time. At times it even fails to find a solution, since there are bottlenecks in the map that don't appear at large search steps. Each of these resolutions constitutes a different execution method for calculating a path, alternative methods which trade-off cognitive-time for quality of solution.

The cognitive scheduler controls the overall distribution of cognitive processing time across all actions. At each time step, it selects the next action from the timeline to execute, chooses an execution method for the action, and executes it. Thus, the scheduler is key to controlling the responsiveness of the cognitive component to real-time constraints. The current version of the scheduler for Phoenix is rudimentary and considers only a short horizon for scheduling decisions. It selects the next action for execution based on timeline ordering, action priority and the amount of time an action has been waiting for execution. A more sophisticated scheduler is being designed now.

4.2.2 Sophisticated Monitoring Through Envelopes

Just as we can explicitly represent the movements of an agent through its physical environment, so can we represent its movement through spaces bounded by failure or other important events. These spaces are called *envelopes*. Typically, one dimension of an envelope is time, and the others are measures of progress.

For example, imagine you have one hour to reach a point five miles away, and your maximum speed is 5 mph. If your speed drops below its maximum, for even a moment, you fail. As long as you maintain your maximum speed, you are *within your envelope*. The instant your speed drops below 5 mph, you *lose* or *violate* your envelope. This envelope is *narrow*, because it will not accomodate a range of behavior: any deviation from 5 mph is intolerable. Most problems have wider envelopes. Indeed, real time systems should be designed to ensure that narrow envelopes are the exception, not the rule.

The following problem illustrates a wider envelope. A bulldozer has one hour to travel five miles, as before, but its maximum speed is 10 mph. It starts slowly (perhaps the terrain is worse than expected). After 40 minutes it has travelled just two miles. It can still achieve its goal, but only by travelling at nearly maximum speed.

Clearly, if the agent waits 40 minutes to assess its progress, it has waited too long, because an heroic effort will be required to achieve its goal. In Phoenix, agents check their envelopes at regular intervals, hoping to catch problems before they get out of hand. One near-term research goal is to develop a theory of envelopes that will tell us when and how often they should be checked.

Agents check *failure* envelopes, which tell them whether they will absolutely fail to achieve their goals, and *warning* envelopes, which tell them that they are in jeopardy of failure. Typically, there is just one failure envelope but many possible warning envelopes. To continue the previous example, the bulldozer would violate a warning envelope if its average speed drops below 5 mph, because this is the speed it must maintain to achieve its goal. Violating this envelope says, "You can still achieve your goal, but only by doing better than you have up to this point." These concepts are illustrated in Figure 3 . The failure envelope is a line from "30 minutes" to "five miles," since the bulldozer can achieve its goal as long as it has at least 30 minutes to travel five miles. The average speed warning envelope is a line from the origin to the goal, but the bulldozer violated that envelope immediately by travelling at an average speed of 3 mph. In fact, it moved perilously close to its failure envelope. The box in the upper right of Figure 3 illustrates that the agent can construct another envelope from any point in its progress. In this example, the new envelope is extremely narrow.

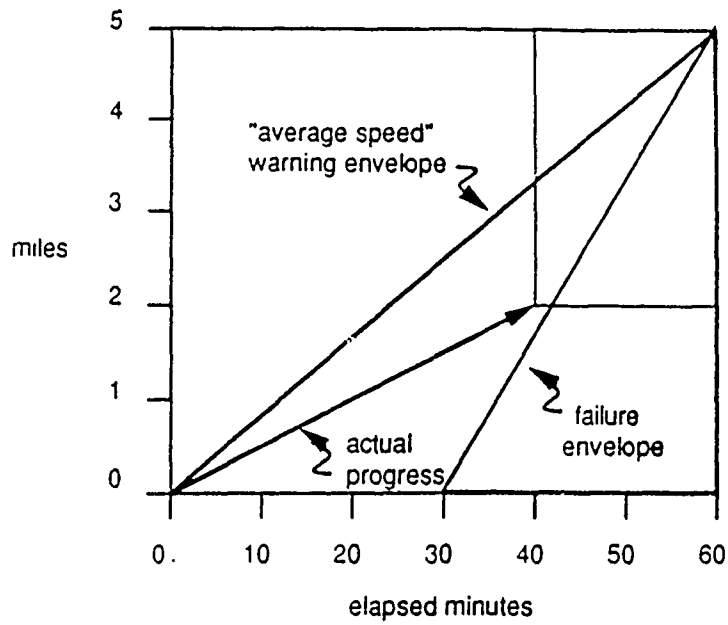


Figure 3. Depicting actual and projected progress with respect to envelopes

Agent Envelopes and Plan Envelopes. We distinguish between the envelopes of individual agents and those of multi-agent plans. In Phoenix, plan envelopes are maintained by the fireboss agent, who coordinates several subordinate bulldozers. Because the environment changes, global plans may be put in jeopardy even if agents are making progress that, from their local perspective, is well within their envelopes. Figure 4 illustrates plan envelopes as they are currently implemented in Phoenix: The leftmost illustration represents the current state of the fire, its projected boundaries after one and two hours, and the firelines that three bulldozers are expected to cut. By projecting where the fire will be, then adding some slack time, the fireboss anticipates that the last of these lines will be cut an hour before the fire reaches it. On the right of Figure 4, we see the actual progress of the fire: After one hour, it has grown less than expected, so the amount of slack time grows (bottom of Figure 4) and the plan stays well within its one hour slack time envelope. But during the next hour, the fire grows more rapidly than expected; so rapidly, in fact, that the slack time envelope is violated. Sometime during this interval, the fireboss will check the plan envelope and discover that it is violated. It then replans and typically sends one or more additional bulldozers to help out.

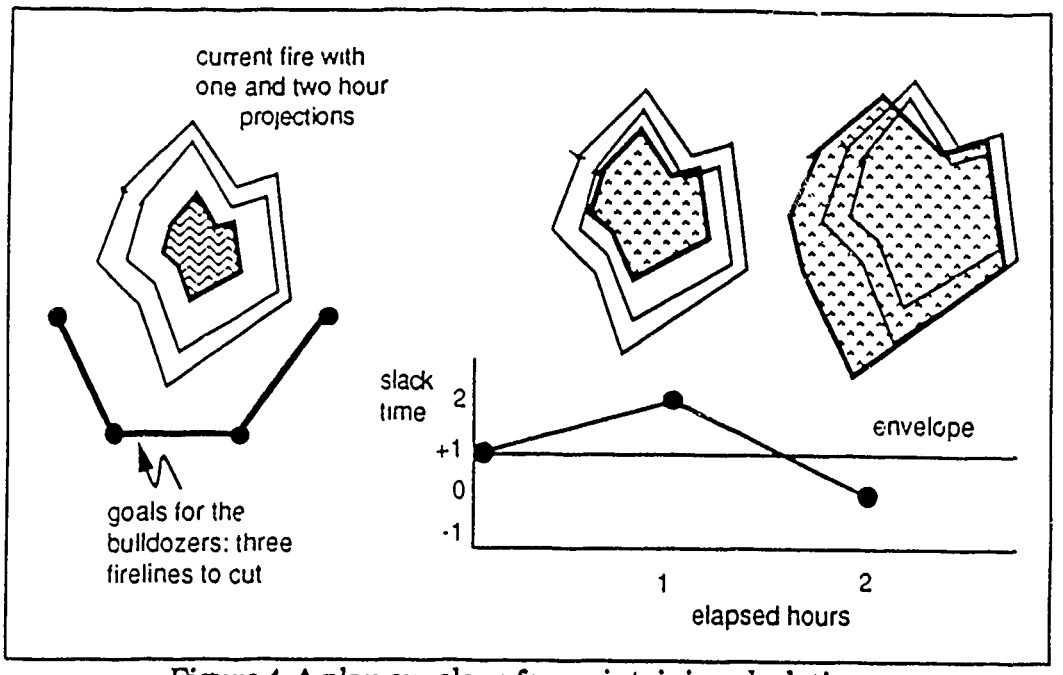


Figure 4. A plan envelope for maintaining slack time.

The Utility of Envelopes. A planner can represent the progress of its plan by transitions within the plan's envelopes. Progress, failures and potential failures are clearly seen from one's position with respect to envelopes, whereas this information is not always apparent from one's position in the environment.

Envelopes function as early warning devices in two ways. First, explicit warning envelopes alert the planner to developing problems. Second, failure envelopes can tell an agent it has failed long before its allocated time has elapsed. In Figure 3, for example, the agent knows it has failed as soon as it crosses the envelope. A third kind of early warning has yet to be implemented: Just as a planner can project the course of events in its environment, so it can project its progress within its envelope and, particularly, when an envelope might be violated. A simple projection method is extrapolation. For example, if we checked the envelope in Figure 4 after 75 minutes we would see a "downward" trend. By linear extrapolation we could estimate when the envelope would be violated. Of course, the downward trend may reverse, or level out. But sometimes it will be worthwhile to have the projected time of envelope violation despite its uncertainty.

Envelopes integrate agents at different levels of a command hierarchy: A fireboss agent formulates a goal and a corresponding envelope, and gives them to a subordinate bulldozer agent with the following instructions: "Here is the goal I want you to achieve. I don't care how you do it, and I don't want to hear from you unless you achieve the goal or violate the envelope." The bulldozer then works independently, not monitored by the fireboss. It figures out where to go, how to avoid obstacles, and how to keep clear of the fire, until its goal is achieved or its envelope violated. Meanwhile, the fireboss is free to think about other agents, other

goals, or to replan if necessary. Envelopes grant subordinate agents a kind of autonomy, and grant superordinate agents the opportunity to ignore their subordinates until envelopes are violated.

We have yet to develop cognitive scheduling mechanisms to take full advantage of envelopes. The design of these mechanisms is motivated by the following questions: How often should envelopes be checked? Should we adopt a fixed interval or a dynamic one, and if the latter, what execution methods will determine when to check next? When should agents project envelope violations and how should they use the projections? Given that checking a plan envelope, or projecting progress with respect to it, may involve collecting and integrating information from the environment and all the participating agents, the cognitive overhead of of these activities can be considerable and must be carefully scheduled.

5. Methodological Issues.

Our overriding research goal is to develop a sound basis for the design of AI agents. AI is a kind of design. We don't design graphics, or VLSI circuits, or mechanical devices: we design intelligent agents. The agents are evaluated by how they behave. Their behavior is determined by their environments and their architectures. Once we adopt this view, we see immediately that we do not know enough about the relationships between agent architectures, behaviors, and environments (the corners of the behavioral ecology triangle in Fig. 1) to design intelligent agents in a principled way. For example, we cannot even precisely define the characteristics of environments (Sec. 2.2), much less behaviors. And we cannot answer the question, "How would the behavior of this AI program, in this environment, change if you change its architecture this way: ... ?" But until we can answer this question, AI system design will remain ad hoc.

In fact, design is one of six research activities implied by the behavioral ecology model. Here is the complete list:

Prediction: How will behavior be affected by changing the architecture of the agent or its environment? For example, how will behavior be affected by changing the size of short-term memory, or by changing the mechanism by which long term memory is accessed? How will behavior be affected if the environment "speeds up," so that events that took N seconds now take N/2 seconds?

Explanation: Why does a particular behavior (presumably unexpected) emerge from the interaction between an agent and its environment? For example, why does an agent that combines long-term, goal-directed behavior with short-term reactive behavior sometimes exhibit something like an approach-avoidance conflict---dashing first toward a goal, then away from it, but getting nowhere in the long run?

Design. What architectures will produce a particular set of behaviors in particular environments? For example, what architectures will enable an agent to respond to events in the environment that occur at very different time scales?

Environment analysis: What aspects of the environment most constrain agent design? What is our model of the environment?

Generalization: Whenever we predict the behavior of one agent in one environment, we should ideally be predicting similar behaviors for agents with related architectures in related environments. In other words, our theories should generalize over architectures, environmental conditions, tasks, and behaviors.

Functional relationships: What knowledge do we need to answer questions in these classes? What are the functional relationships between the architecture of an agent and its behavior?

Both the behavioral ecology model and the S/E model of Rosenschein, Hayes-Roth, and Erman (see their paper in this volume) explicitly acknowledge the relationships between architecture the environment, and behavior. Rosenschein et al. denote the architecture and environment S and E, respectively; and characterize behavior as a sequence of state changes called a run. Furthermore, Rosenschein et al. seem to implicitly subsume, in what they call measurement and evaluation, some of the research activities above. But because neither the S/E model nor the behavioral ecology model make predictions, it is premature to compare them except to note some apparent differences in emphasis.

Rosenschein et al. view the "S/E boundary" as flexible, so that sometimes the environment can be made responsible for an activity that, in other circumstances, we might require of the agent. For example, with the general vision problem currently unsolved, we might construct an environment that "preprocesses" sensory data for the agent, thus moving the S/E boundary inward, toward the agent, bypassing the need for sophisticated sensors. This example suggests a small apparent difference between the S/E model and the behavioral ecology model: whereas the S/E model seems to assume a simulated environment, the behavioral ecology model does not. Although the Phoenix project uses a simulated environment, our principal research tasks (prediction, design, explanation, etc.) do not *presume* a simulated environment. It isn't clear yet whether the principal research tasks of Rosenschein et al. presume a simulated environment.

This raises the methodological question of whether one should use simulations at all. Some researchers insist that the subtleties of real environments are "lost in translation" to simulated environments. This is to some extent a straw man, because we don't view simulations as accurate representations of the real world. (In fact, we recently got into trouble by claiming that the Phoenix environment is an accurate simulation of forest fires.[4]) But it is important to distinguish *realism*

and *accuracy*. Realism is necessary for our research; accuracy is not. Here are some examples of the distinction: In a realistic simulation, processes become uncontrollable after a period of time; in an accurate simulation, the period of time is the same as it is in the real world. In a realistic simulation, agents have limited fields of view; in an accurate simulation, agents' fields of view are the same as they are in the real world. In a realistic simulation, the probabilities of environmental events such as wind shifts are summarized by statistical distributions; in an accurate simulation, the distributions are compiled from real-world data. When possible, we use accurate data; for example, in Phoenix we use Defense Mapping Agency data of elevation, ground cover, and so on, and the fire dynamics are derived from U.S. Forest Service manuals (NWCG Fireline Handbook, 1985). But the goal of our research is not to accurately simulate forest fires in Yellowstone National Park. It is to understand the design requirements of agents in realistic environments—environments in which processes get out of hand, resources are limited, time passes, and information is sometimes noisy and limited.

With this in mind, we see that simulations have several advantages:

Control. Simulators are highly parameterized, so we can experiment with many environments. For example, we can change the rate at which wind direction shifts, or speed up the rate at which fire burns, to test the robustness of real-time planning mechanisms. Most important, from the standpoint of our work on real-time planning, is the fact that we can manipulate the amount of time an agent is allowed to think, relative to the rate at which the environment changes, thus exerting (or decreasing) the time pressure on the agent.

Repeatability. We can guarantee identical initial conditions from one "run" to the next; we can "play back" some histories of environmental conditions exactly, while selectively changing others.

Replication. Simulators are portable, and so enable replications and extensions of experiments at different laboratories. They enable direct comparisons of results, which would otherwise depend on uncertain parallels between the environments in which the results were collected.

Variety. Simulators allow us to create environments that don't occur naturally, or that aren't accessible or observable.

Interfaces. We can construct interfaces to the simulator that allow us to defer questions we'd have to address if our agents interacted with the physical world., such as the vision problem. We can also construct interfaces to show things that aren't easily observed in the physical world; for example, we can show the different views that agents have of the fire, their radius of view, their destinations, the paths they are trying to follow, and so on. The Phoenix environment graphics make it easy to see what agents are doing and why.

Let us return now to the comparison of the S/E and behavioral ecology models. We noted that the former model represents behavior as "runs," sequences of state transitions (or as measures over runs), whereas the behavioral ecology model is

inspecific about how to represent behavior. On the other hand, the behavioral ecology model is quite specific about the causal relationships that hold among the environment, the agent architecture, and the agent's behavior. The behavioral ecology model comes from biology; it regards the architecture as analogous to the genotype and the behavior as analogous to the phenotype. And it assumes that selection operates on the phenotype. Thus, there is no direct causal link between an agent's environment and its architecture; rather, the environment ensures that behaviors are differentially rewarded, so the architecture must be modified to produce "good" behaviors. This, then, is what we mean by a good architecture—one that produces behaviors that are good in a particular environment.

It's important to know whether such behaviors can be generated by design, that is, by intentional modifications to the architecture, or whether they must evolve by search. Advocates of emergent behavior often take the latter view. They say that one cannot generate the phenotype from the genotype; one cannot predict how a moderately complex architecture will behave. This has important practical and methodological implications for IRTPS. Do we build IRTPS systems "top down," by assembling components that are predicted to behave in particular ways, and damn the emergent behaviors? Or do we build them "bottom up," by assembling components incrementally and empirically, waiting for desired (and undesirable) behaviors to emerge? In fact, we mix the approaches in proportions determined by the degree to which behaviors can be predicted from architectures (or components of architectures). Moreover, this degree of predictability is determined in part by the desired precision or scale of the predictions. If you want to know the precise number of cpu seconds that a process will run, you are probably out of luck. But if you want to know the upper bound runtime, it may be possible. You probably can't know the exact location of a fire ten minutes from now, but you can certainly draw a circle that has a high probability of circumscribing the fire. Thus, the question of whether behaviors can be generated by design depends intimately on how precisely we want to specify and predict the behaviors.

This brings us to a final methodological issue: evaluation. Let us first ask, What is being evaluated? Whether an architecture exhibits "timely" behavior? or exhibits a good tradeoff among several desired behaviors? Whether the behaviors are exhibited in a sufficiently wide range of environments? Whether we can predict when the behaviors will and will not occur? Whether we understand the functional relationships between architecture and behavior well enough to design an agent that will exhibit desired behaviors in a new environment? All of these should be evaluated. More pointedly, evaluation *cannot* stop with the demonstration that a system "works," however sophisticated the demonstration! We must take at least two more steps: We must attempt to show *why* the solution works (or doesn't work). This is uncommon, but essential if we are to make progress as an engineering field. The third step is to show why any solution with such-and-such abstract characteristics must work (or not work). This requires models of the behaviors and environments under study.

5.1 An Example of Design for IRTPS

We will briefly illustrate the previous points, and the terminology in Section 2.2, with the example of the design of Phoenix's cognitive scheduler. It's current cognitive scheduler is very weak. We are designing another one that achieves many of the behavioral goals of IRTPS. In the terms of Section 2.2, this seems to imply that we should list the problems and inherent problems, describe the relevent ECs and ACs, and after analyzing how the problems arise out of the interactions between ECs and ACs, we propose solutions and solution realizations. In fact, this seems to be an idealization. Instead we start with an *informal* description of some problems, and then hunt around for ECs and ACs that we believe account for the problems. The result is a formal description of the problems in terms of ECs and ACs. Then we generate solutions and solution realizations.

Here is an example of the first steps.

Informal description: The plan selection mechanism may take too long to find a plan. As a result, the fire may burn too much area, or may become uncontrollable. (Note that this is intentionally vague, to show how we formalize the problem description in terms of ECs and ACs.)

Environment characteristics: What is going on in the environment that could contribute to the problem, as informally described above? Let's concentrate on one thing, the spread of the fire. We want to model this in a way that allows us to firm up the informal problem description. Suppose we model the spread of the fire as an exponential process analogous to compound interest and population growth. Then, the perimeter of a fire after t time units is:

$$p = p_i(1 + r)^t$$

Here, p is the perimeter of the fire, p_i is the initial perimeter of the fire, r is the percentage increase in the fire perimeter every time unit, and t is the number of time units that have elapsed.

Architecture characteristics. For now, we will list just two characteristics: It takes a period of time, d, to generate a plan and get the bulldozers to the fire to begin implementing the plan. And once at the fireline, bulldozers dig at a constant rate.These are both oversimplifications, but useful, as we shall see.

Now we can say more formally what the problem is:

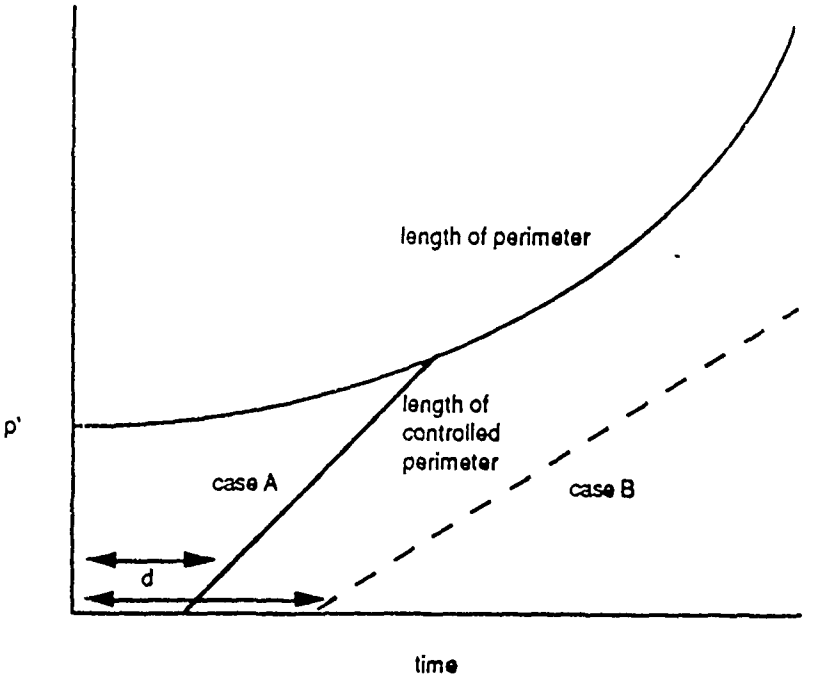


Figure 5

The curved line represents p, the length of the perimeter, as a function of time. It begins not at the origin, but at a point that represents the initial perimeter of the fire (e.g., its size when detected). We assume for simplicity that the steepness of the curve is described by one parameter, r, which captures factors such as wind speed and fuel type. Obviously, a more complex model could be generated if needed. After some delay, d, a plan is detected and some bulldozers are dispatched and then arrive at the fire. They begin cutting fireline at a constant rate, so the length of the controlled perimeter increases at a linear rate determined by the number of bulldozers. We show two possibilities, case A and case B.. In case A (solid line) the bulldozers arrive at the fire sooner, and in greater numbers than in case B (dashed line). In fact, in case A the fire is controlled, whereas in case B it is not. We know this because the line for case A intersects the line for the perimeter, which means that at some point, the length of the controlled perimeter equals the length of the fire perimeter; or, all the fire perimeter is controlled. In case B, this doesn't happen.

Before we can rephrase the informal problem description more precisely, we need to know what affects the parameters represented in the diagram above. This will tell us what we control as designers, what the system itself controls as an autonomous agent, and what the environment alone controls.

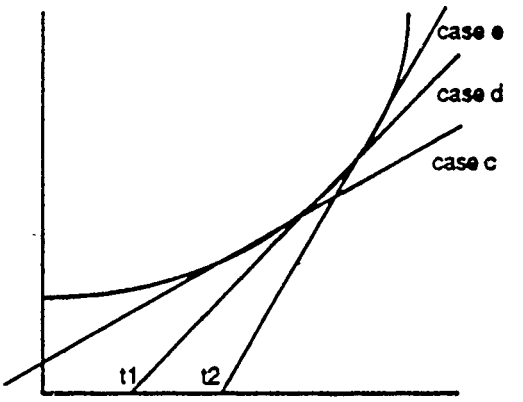
p' — This could be reduced if the fire was sighted earlier. The lower limit on the speed with which the fire is sighted is an AC that we control. It depends on things like how big a fire must be before it is noticed, how often the watchtowers look, how long it takes them to report their findings, how long it takes the fireboss to notice, etc. Most of these ACs have lower limits that we control, and actual values that the agent controls.

r — This parameter, which determines the steepness with which the perimeter increases, is an EC.

d — as with p',we control the lower limit on d, and the actual value is controlled by the agent.

slopes of "controlled perimeter" lines — this has an upper limit that we control (by controlling the number of available bulldozers) and an actual value that the agent controls, by controlling the number of bulldozers committed to the fire. .

At this point, we can begin to give formal descriptions to problems. For example, what is a *deadline*? In general, a deadline is a point at which the value of problem solving changes, usually downward. Consider a hard deadline for the plan selection process. In the previous diagram, this is represented as an upper limit on d. Consider three cases, denoted c, d, and e in the following diagram:



In case c, the slope of the "controlled perimeter" line is shallow because, say, it corresponds to a plan that involves only two bulldozers. Moreover, the deadline for the institution of the plan *has already passed*. You can see this by shifting the line for case c to be tangent to the "perimeter" line. By the same operation you can see that, for case d to succeed, the bulldozers (more of them than in case c, hence the steeper slope) must be at work before t1; and for case e to work, they must be busy by t2. Any delay longer than t1 or t2 shifts the respective lines to the right, ensuring that they will never intersect the perimeter line and the fire will never be controlled.

Now we can be more precise about the problem: For given values of r and p' (assuming the fire has just been sighted), find a plan that is expected to contain the fire and that can be instituted before its deadline.

Note that the original problem, a failure to get plans ready in time, has been formalized in the context of an agent model and an environment model. Moreover, a common IRTPS term has been defined in these contexts. One might argue that, in the process, we have taken a nice, general term like "deadline" and replaced it with something that is so specific to Phoenix as to be unusable. We believe we have done exactly the opposite. Not only have we made a vague term precise, but we have also identified a very general functional relationship or "rule" associated with the term: Imagine that the perimeter of the fire grows linearly, not exponentially. Then the notion of deadline illustrated in Figure 5 would not exist. If a process F grows linearly, and another linearly-growing process B is trying to control it, then a comparison of the growth rates of F and B will tell us whether B will succeed, and when it will succeed (assuming the growth rates don't change). If B grows faster than F, then it will control F eventually. The only effect of delaying the onset of B is to delay the control of F. On the other hand, if F grows superlinearly, as in Figure 5, and B grows linearly, then a delay does not merely delay the event in which B controls F, it may make that event impossible (as shown by the dashed line in Figure 5). we believe this is a very general phenomenon, and thus a very general interpretation of "deadline": A deadline is the point at which a linear process becomes incapable of catching—at any time in the future—a superlinear one. Obviously this can be generalized to functions of other orders—a sublinear process trying to catch a linear one, and so on.

As we evaluate the Phoenix project, we will certainly ask whether it plans in a timely way, whether it meets deadlines, balances cognitive load, exhibits graceful degradation, and so on. But the most telling evaluation will be whether we have been able to engage in the specialization-generalization process illustrated above: Whether we gave terms such as "deadline" precise interpretations in terms of Phoenix ECs and ACs and then generalized them again, as we did when we said a deadline is a point at which one process becomes incapable of catching another.

References:

Cohen, P. R., Greenberg, M. L., Hart, D.M., Howe, A. E. 1989.Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. AI Magazine, Vol. 10, No. 3. pp. 32-48. Fall, 1989.

Cohen, P.R. and Howe, A. E. 1988. How Evaluation Guides AI Research. AI Magazine 9(4) 35—43.

NWCG Fireline Handbook 410-1, National Wildfire Coordinating Group, Boise, Idaho, Nov 1985.

Rosenschein, S. J., Hayes-Roth, B., and Erman, L. D. Notes on Methodologies for Evaluating IRTPS Systems. In this volume.

X. MICE: A Flexible Testbed for Intelligent Coordination Experiments

Edmund H. Durfee and Thomas A. Montgomery
Artificial Intelligence Laboratory
Dept. of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48103
durfee@caen.engin.umich.edu

Reprinted from Proceedings of 9th Workshop on Distributed Artificial Intelligence. Sep., 1989.

1 Introduction

Research in distributed problem solving has investigated several application domains, such as air traffic control [11], distributed vehicle monitoring [8,10], and factory floor control [9]. Each application emphasizes certain coordination issues. For example, air traffic control emphasizes global coordination to guarantee the avoidance of aircraft collisions, distributed vehicle monitoring emphasizes the generation and sharing of tentative (possibly incorrect) partial results to converge on global solutions, and factory floor control emphasizes the proper allocation and scheduling of production tasks at different processing sites. Because of their different emphases, techniques for coordination that have been developed for one application have often been inappropriate for another application. Moreover, transferring techniques between applications has proven difficult because of the knowledge engineering and translation needed to recode a technique in a new domain. Thus, distributed problem solving research has been limited by the inability to easily evaluate different coordination techniques in a variety of application environments.

To overcome this limitation, we have developed a flexible testbed called MICE (Michigan Intelligent Coordination Experiment) that extends the ICE testbed developed at the University of Southern California [6] in which artificially intelligent agents interact on a two-dimensional grid [1,4]. MICE retains this two-dimensional grid model of the world and adds a number of extensions that allow greater flexibility in the coordination issues that can be presented to the agents. MICE provides an environment where agents "live," and imposes constraints on the capabilities and actions of agents and on the interactions between agents. These constraints affect the mobility of agents; the range, accuracy, and time needs of their sensors; their ability to move, create or remove other agents; and how collisions or other spatial relationships affect agents.

Our initial goal has been to build MICE as an experimental testbed that does not simulate any specific application domain, but can instead be modified to impose a variety of constraints on how agents act and interact so that we can emulate the different coordination issues that arise in various application domains. Thus, rather than building several testbeds and reimplementing the agents' reasoning architectures and coordination techniques for each, we can leave the agents alone and instead modify the parameters of our single testbed to simulate the important coordination characteristics of different domains. An added benefit of this approach is that, because our testbed does not have to fully simulate any application domain (only the domain's coordination issues), we reduce the knowledge engineering effort for building the testbed, and we decrease the chances of solving coordination issues with approaches that are specific to an application domain.

In this paper, we describe the motivations behind MICE by presenting background information and by outlining the approach taken in its design. We then describe the capabilities of the MICE system, and illustrate how it simulates different environments in which AI systems should coordinate. We conclude with plans and directions for the future.

2 Background

One objective of developing generic DAI testbeds is to simplify the investigation and evaluation of coordination mechanisms in a variety of contexts, in an attempt to discover general principles of coordination. MICE differs from previous generic DAI testbeds because of the minimal assumptions it makes about how agents are implemented. For example, the MACE [5] testbed provides a language for defining both agents and their environments, and provides many facilities for monitoring, error handling, and interacting with the user. This language simplifies the task of defining new types of agents, but also limits the agents that can be built and evaluated to those that are specifiable in the MACE language. In contrast, the MICE testbed provides facilities for describing only the environment in which agents act and interact, leaving the user with the flexibility to implement agents in any way desired (so long as they can interface with Lisp). The increased latitude in how MICE agents can be implemented places a greater burden on the agents' developers, but allows experimentation with widely different architectures, including blackboard systems [3] and Soar [7]. Because MICE specifies only how agents interact *indirectly* through the environment, the agents' developers are free to specify how agents interact *directly* through communication. Experimenters can populate a MICE environment with agents that have different architectures provided they define communication protocols between these agents.

3 Design

The research we are conducting with the MICE system is based on two very important hypotheses.

1. Coordination issues arise from a combination of the requirements of an application environment and the capabilities of the agents that inhabit the environment. For example, the catastrophic consequences of collisions in air traffic control constrain how vehicles behave, but the difficult coordination issues triggered by these constraints also depend on the vehicles themselves, such as how much each can know about the environment and about the others, how quickly they can reason, and what their communication capabilities are.

2. Coordination does not so much depend on "techniques" that are "given" to agents, but, instead, that reasoning about coordination is a fundamental aspect of intelligence that should permeate the reasoning of an agent. For example, coordination is not achieved by starting with an artificially intelligent system and then giving it a coordination technique such as the Contract Net protocol [10]. Such an approach overlooks the fact that, to intelligently use this protocol, agents must internally reason about coordination. Deciding how and when to decompose, announce, award, and bid on tasks using the protocol is difficult and can require sophisticated reasoning about local and non-local goals, plans and constraints. The ability to coordinate should be designed into an agent's

reasoning architecture, so studying different ways of coordinating might means studying alternative architectures.

By investigating alternative architectures under a variety of environmental constraints, we can begin to identify the general coordination principles that must be considered in any truly versatile architecture for intelligent coordination. To facilitate this investigation, we have designed MICE to meet the goals of:

1. Flexibility. MICE can easily model the coordination issues that arise in different application domains.

2. Limiting knowledge engineering requirements in simulating new scenarios. A library of domain-specific predicates allows researchers to build environments with unique combinations of coordination issues.

3. Providing a clean interface to the intelligent agents. The interface between MICE and the agents is well-structured to give researchers the latitude to implement agents in any reasoning architecture desired.

4. Helping researchers collect the results of running coordination experiments. MICE provides a set of tools that can be used to view the interactions between agents in the environment, to review previous states and events, and to collect statistics over the course of experimental runs.

MICE is intended to model the significant coordination issues of application domains without modeling domain details that do not directly impact coordination. This distillation from the space of completely detailed domains to the coordination issues that they present allows us to use MICE to combine coordination issues in unique ways. The ease with which new sets of coordination issues can be combined and presented to intelligent agents, without forcing an interface change on the agents, facilitates the evaluation of how well the agents coordinate.

MICE has been designed to maximize its portability as well as the ease with which it can reproduce experimental results. To provide these features, MICE is implemented in a standard language, Common Lisp, and runs on a serial processor, currently a Texas Instruments Explorer. These choices provide greater portability over the use of special-purpose languages and multi-processor machines, both of which are not as readily available. By simulating concurrency on a serial processor, MICE retains greater control over the actions of agents, making it possible to exactly reproduce experimental results. Furthermore, since MICE uses simulated time to model concurrent agent actions, agents are not subjected to real-time constraints. This allows us to investigate reasoning architectures that have not yet achieved real-time operating speeds.

4 Implementation

In this section we describe the MICE system and the options it provides in more detail. During the course of an experimental run MICE maintains a state history

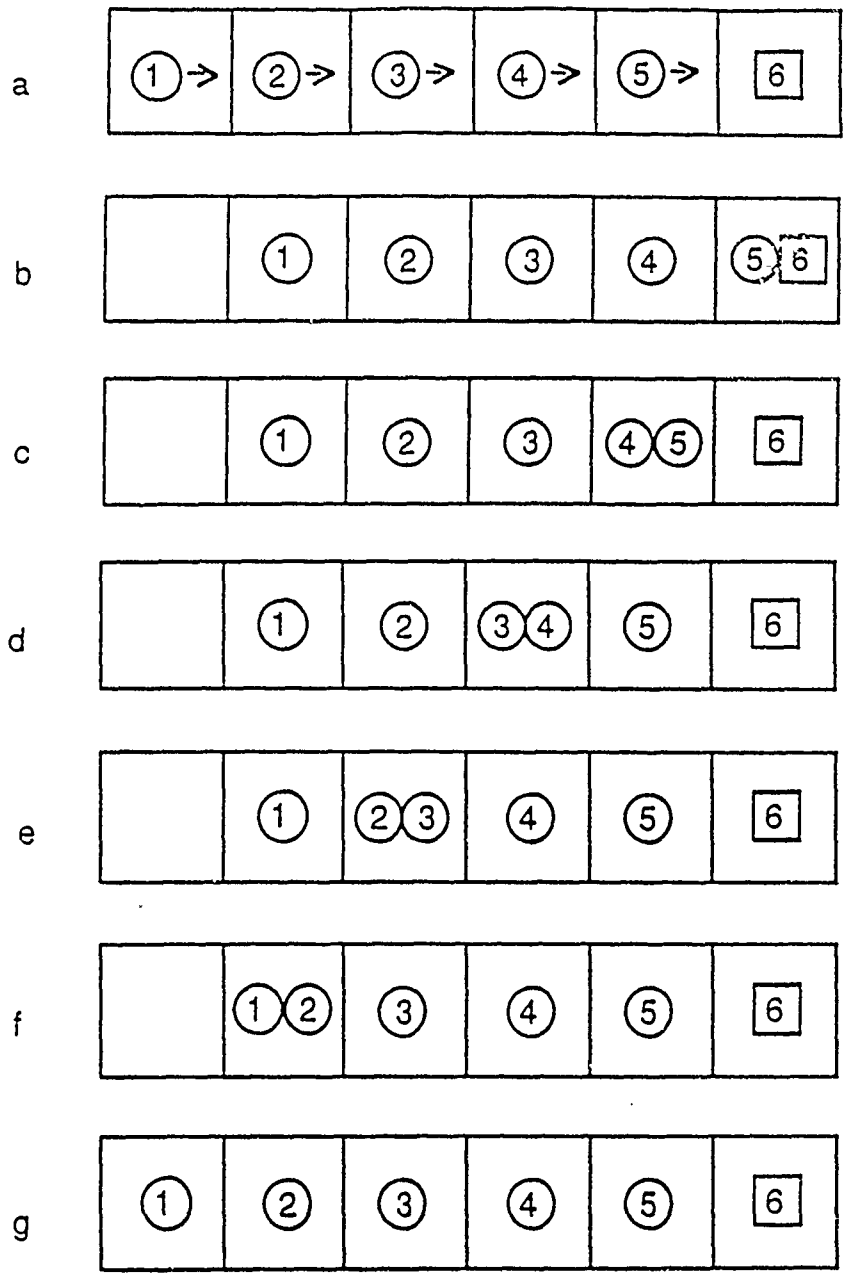
that includes the location and orientation of each agent, the status of each agent (created, activated, deactivated or removed), actions taken or attempted by each agent, and characteristics of locations in the grid. This state information can be saved away into a file for further examination after the run is complete.

In simulating a single time step, MICE first allows the intelligent agents to decide what actions they wish to perform. The actions that they can choose from are: moving; changing their orientation; scanning for other agents and for features of the grid; linking to other agents; unlinking; making status changes by creating new agents, removing existing agents, activating, and deactivating. Agents can also specify an amount of time spent "reasoning". This allows the agents to associate time costs in the simulated environment with the actual time they have spent in computation or in communication with each other.

After all agents have chosen actions to perform, MICE executes the actions in the environment to produce a new state. Any conflicts in this state are resolved through a combination of fixed and user–specifiable predicates that ensure a consistent resulting state. An example of a conflict that must be resolved is when two agents that are not allowed to overlap (occupy the same location) decide to move to the same location. MICE will detect this situation and call the resolution predicate that has been chosen for the experimental run. One option causes the agents to bounce off of each other, returning them to their prior locations. Another choice is to have an authority relationship such that the agent with higher authority gets the location and the lower authority agent is pushed back. The resolution process gets complicated when resolving one location causes a new conflict in another location. This can happen, for example, when a number of agents that are not allowed to overlap are following each other. If the lead agent attempts a move into an illegal location and is moved back, it will now be in conflict with the agent that was behind it. This results in a domino–effect of undoing moves back through the line of agents (Figure 1). In the worst case scenario, our resolution procedures will result in all actions being undone, returning the environment to its previous state. That previous state is guaranteed to be consistent since it was resolved at the end of the last time step. Under no circumstances are actions undone that were completed in a prior time step.

MICE uses graphics to display the environment's state at each time step. Agents and significant grid features are represented by geometric icons (squares, circles, triangles, etc.). These icons can be changed in response to the occurrence of events. For example, an agent represented by a filled square may change its representation to a hollow square when it deactivates. MICE has an event-driven set of predicates that can be used to aid interpretation of an experimental run. Such predicates can be used to maintain statistics, change the graphic display of an agent or a grid location, or cause other changes in the environment. In a fire-fighting scenario, for example, after an area has been burned, its characteristics can be changed so that fire cannot move through it again. At the same time, statistics can be updated on the amount of area consumed by the fire, and the graphic representation of the burned location changed to reflect its condition.

The flexibility of MICE is evident in the parameters that can be specified to create new environments to test the skills of intelligent agents. These include:



At one time, agents 1 through 5 all decide to move east (a). MICE resolves the resulting conflict between agents 5 and 6 (b) by moving agent 5 to its previous location. This results in a new conflict between agents 4 and 5 (c). Successive iterations of MICE's conflict resolution procedure (d, e, f) finally produce the consistent state (g). The net effect of the resolution is a return of the environment to the state it was in at the previous time step.

Figure 1: A Chain Reaction in the Resolution of Agent Moves

- The resolution of collisions between agents. When agents collide by attempting to move through each other or by moving to the same location, their final positions can be resolved in a number of ways. For example, they may pass through each other, bounce off of each other or push each other.

- The effects of collisions. Agent characteristics can change as the result of a collision. For example, a collision might cause an agent to lose (or gain) resources, influencing its future capabilities, such as when it loses energy in a collision and afterward moves more slowly.

- The effects of being in certain spatial relationships. Any detectable spatial relationship can trigger a change to an agent's characteristics. For example, in the predator-prey environment (Section 5.1, prey agents that have been surrounded by predator agents are removed from the environment.

- The effects of other events or actions in the environment. Predicates can be associated with any event such as agent movement, linking, unlinking, status changes, etc.. For example, if an agent simulates "picking up" another agent by linking to it, a consequence might be the other agent becoming deactivated so that it would no longer be capable of moving itself.

- Agent characteristics. These include the agents' starting locations, the time costs of moving and of scanning, which agents obstruct movement and sensing, and so on.

- Features of the grid and locations in the grid. The size of the grid can be specified as can features of individual locations. For example, we can assign attributes to a location to block movement through the location, or obstruct sensing beyond it.

- The entry into the code for the intelligent agents. When an agent must make a decision on what to do next, MICE has a pointer to the agent's *invocation function*. By calling this function, MICE passes control to that agent's reasoning process. We have implemented intelligent agents directly in Lisp and in a blackboard system. We also have a human interface that gives the user a menu-driven choice of actions.

- Termination predicate. The termination predicate decides when an experimental run is complete and outputs results of the run. For example, in the fire-fighting scenario, the run is complete when there are no active fire agents (simulating the fire being extinguished or burning itself out). At the end of the run, the termination predicate displays statistics that give indications as to how well the fire was fought.

5 Example Simulations

MICE provides us with a flexible framework in which we can simulate a wide variety of environments that involve multi-agent coordination. That is, MICE allows us to simulate the issues in coordination that arise in a wide variety of domains, and we do not simulate other details of domains that have no bearing on coordination. To simulate a particular multi-agent environment, we must give MICE two kinds of information. First, we must specify the environmental parameters, predicates, and constraints; MICE uses these to calculate the ramifications of the set of actions taken by the agents at any given time. Second, we must define functions to invoke for an agent when that agent is to decide on its next action. These two kinds of information are tightly intertwined, because agents' decisions might (and usually should) involve some knowledge about their shared environment. To illustrate how we use MICE to simulate particular multi-agent environments and populate these with agents, we describe several simulations that we have implemented in MICE using 20 by 20 grids.

5.1 Predator-Prey

The inspiration for the MICE environment sprang from previous work that simulates the interactions between predators and prey in a two-dimensional grid environment [1,4,6]. Although different implementations have all concentrated on the problem of how agents of one type (predators) can surround and capture agents of a second type (prey), the constraints and capabilities of the agents have slightly varied from one implementation to the next. MICE allows us to simulate a wide range of constraints and capabilities for this problem, including:

- An agent's sensing range (how far it can sense), period (how often it can sense), sensitivity (what objects and agents it can sense), and time costs (how long it takes to sense) can be specified. Typically, a predator can be given the ability to continuously sense an area around itself for other predators and prey, while prey have no sensory capabilities.

- An agent's mobility, in terms of which directions it can move and how quickly it can move in each direction, is parameterized. Typically, both predators and prey can move equally quickly and well in any direction.

- Agents' spatial constraints, such as whether two agents can occupy the same location, can be specified, along with a predicate to resolve conflicts. In the predator-prey, we typically do not allow two agents to occupy the same location, and when they attempt to do so the conflict is resolved by moving them back to where they were before they attempted moving.[1]

[1]Some previous simulations of the predator-prey domain allowed only one agent to move at a time in cycles of turn-taking, so that no such conflicts arise. We instead simulate the simultaneous activity of all agents at any given time, so two agents that are unaware of each other's current decision might take conflicting actions.

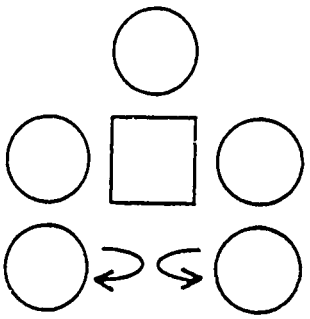
We have implemented several different predator-prey environments, with minor variations on agent goals, capabilities, and constraints. Our most common scenario involves 2 prey agents that are moving across the area to reach a safe area, and 6 predator agents that attempt to capture them before they get there. Because it takes 4 predators to surround a prey, the predators must attack the prey one at a time. To succeed they must coordinate their decisions as to which of the prey they will attack at any given time.

Within this simulated environment, we have done some preliminary evaluation of agents that have been implemented in two different ways. Initially, we wrote some straightforward Lisp code to control agents' activities. In this implementation, a prey agent simply moves in a direction that can reduce the distance between it and the safe area. A predator agent scans the area within its sensor range to develop the most up-to-date view of its vicinity, and if it sees no prey it does not move. If it sees one or more prey agents, it builds goals to occupy each of the 4 adjacent locations to each of the prey, unless it sees that a fellow predator is already in that location. It then moves toward the nearest of its goals.

Not surprisingly, the performance of the predators was generally very poor, unless their initial distribution in the area led to fortuitous coordination. As has been discussed more fully elsewhere [6], entrapping prey can require a great amount of coordination between predators—coordination that our original simple predators were not taking into consideration. As a result, the predators would often fail to block the escape route of prey, or worse yet, the 6 predators would split into 2 teams of 3 and chase separate prey agents with no hope of success. In fact, even when they would be in a good position to capture a prey agent, the lack of coordination between predators could cause them to repeatedly collide with each other so that they would make no progress as a team (Figure 2).

To improve coordination, we have developed a preliminary implementation of agents as separate but communicating blackboard systems. The implementation has been done in GBB [2] using the simple agenda-based control shell, but modifying this shell to simulate a multi-agent environment. When MICE calls an agent's invocation function, this function triggers the continued processing of the agent's blackboard system. The agent's knowledge sources (KSs) can act on the internal reasoning of the agent by affecting its blackboard, and can also return commands to the MICE shell for simulating agent actions. For example, a predator agent has a SCAN KS that sends an appropriate command to the MICE shell, which in turn returns scan information that the KS can process. The small number of externally invocable functions in the MICE shell are available to the GBB-based system, and MICE can invoke a few external GBB functions, so these very modular shells can be used in tandem through a well-specified interface.

To our GBB predators we have improved coordination by adding rudimentary capabilities for communication and for modeling other agents. Essentially, when a predator decides which prey it will chase, it sends messages to the other predators with this information. As predators receive this information, they update their models of each other to reflect this additional knowledge. As these models improve, a predator



The prey (indicated as a square) is surrounded on 3 sides by predators (circles). Two predators are attempting to occupy the last side, but because they lack coordination, they repeatedly collide with each other and bounce back to their initial positions. Unless the predators have some additional coordination capabilities, this situation will lead to a stalemate.

Figure 2: A Stalemate Position for Uncoordinated Predators

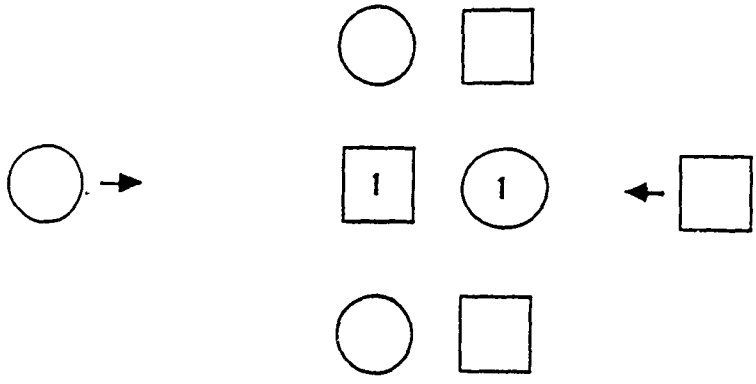
can make better decisions about what prey to chase. For example, if it has a choice between being the fourth predator chasing a distant prey or the only predator chasing a nearby prey, it will give preference to the former. If more than 4 predators are chasing the same prey, they can reason about which should abandon the chase in order to chase other prey (to avoid wasting resources). If the predators have initially divided into teams which are each too small to successfully surround prey, they can use heuristics involving measures such as the center of gravity [1] to decide which teams should be dissolved and which other teams should be enlarged.

Our initial blackboard-based implementation has, not surprisingly, shown significant improvement over the much less sophisticated agents first developed. However, there are still important improvements that we can make to our agents so that the agents can reason even better about themselves, each other, and their environment. For example, if predators could model the goals of the prey (such as moving toward a particular safe area), it would only take 2 predators to block a prey from its goal until other predators arrive to complete the capture. On the other hand, if prey are moving randomly in directions away from nearby predators, then 2 predators will not be sufficient to keep some prey in check (unless the predators can also move more quickly than the prey). Because MICE allows us to easily change the characteristics of the environment and of the agents, we plan to investigate the different—and common—techniques for coordination in different specific instances of the more general predator-prey domain.

5.2 Predator-Predator

Using MICE, it is simple to extend the simulated predator-prey domain to a domain with 2 kinds of predators, each of which preys on the other kind. By modifying a few characteristics of the prey in the predator-prey simulation, we can give the prey the capability to capture predators. In addition, our simulation increases the number of agents (obviously, 2 prey-turned-predators would not have much of a chance since 4 are needed to capture some agent). We have predominantly been experimenting with simulations where there are 12 agents of each type, the agents being randomly distributed in the 20 by 20 grid.

As in the predator-prey environments, one of the coordination tasks is for predators to dynamically team up to capture some prey. However, each predator must also have a goal of avoiding being captured itself. This additional goal can cause major changes in agent behavior, and in team organization. For example, although we have not studied more centralized organizations where one of the team members coordinates and fully controls the other 3 in the capture of some prey [1,6], these are feasible organizations for predator-prey environments. When predators can become prey, however, the centralization of control in some team manager might be less desirable: Rather than depending on some captain (who might might have an incomplete view of the agents it is coordinating) to recognize when an agent might be in danger and telling that agent what to do, we might prefer giving the agent itself some local autonomy so that it can unilaterally decide to quit being a team player (at least temporarily) in order to save its own skin. The tradeoff between the benefits



The circular predator 1 holds its position to help capture square predator 1, even though it will be captured itself sooner.

Figure 3: The Costs of Favoring Cooperation over Local Autonomy

of capturing agents (which encourage cooperation) and the costs of being captured (which fall essentially on individuals) leads to a tension between the simultaneous goals of being cooperative but also retaining some autonomy.

In our experiments to date, we have studied agents that are predominantly cooperative, so that one agent holds its post of occupying one side of another agent to capture even though it is sometimes surrounded and captured first (Figure 3). As a consequence, the agents often come together in a mixed group and jockey for position, attempting to surround each other. These situations often lead to stalemates as agents repeatedly collide with each other and no progress by either side is made. One way that we have used the MICE testbed to explore alternatives within this type of problem is to assign different authorities to the agents, and replace the collision function that returns agents to their previous locations when they collide with a function that allows agents with higher authority to occupy their desired locations, forcing lower priority agents out of their way.[2] Once again, MICE allows such changes to be made easily.

5.3 Forest-Fire Fighting

With a few modifications to our predator-prey environment, we can simulate some coordination issues for a cooperative domain such as forest-fire fighting. As a simple

[2]We also use this function to allow agents to push each other around. For example, we can simulate 2 high-authority robot agents that are moving a (low-authority) box by cooperatively pushing it to some desired location.

example of this domain, we can build fire-fighter agents and fire agents. The fire agents move in certain patterns, and we can specify predicates that simulate fire moving downwind and burning differently in areas with different groundcover (which is simulated by giving appropriate feature information to different locations on the grid). If unchecked for a certain amount of time, a fire agent creates a copy of itself at an adjacent location. Thus, the fire can spread and enlarge over time. Moreover, once a fire agent has occupied a location, the features of that location are modified so that fire cannot spread there in the future. Firefighter agents are simulated in a simple way by specifying an initial firefighting capability for each. When a firefighter encounters a fire agent, it applies itself to destroy the fire agent, but as a result it has less capability (it is weakened).

The firefighter agents must work together to contain and extinguish the entire fire before exhausting their capabilities. Strategic considerations include surrounding the fire to contain it, fighting it before it can spread, and concentrating on the fire's front. However, because each firefighter might have a limited local view of the fire (limited sensor range), the agents might have different perceptions as to how to pursue these strategic goals. The agents must therefore communicate and coordinate their actions to work as an effective team.

To date, our firefighting agents use very simple knowledge sources and coordinate little to fight the fire. Although the firefighting agents have some commonality to the predator agents developed for other environments (such as the goal of surrounding and containing other agents), the characteristics of the simulated firefighting environment, including the ability of fire agents to generate new fire agents and to move at different speeds in different areas, bring in different issues in reasoning about local and coordinated actions. Moreover, in MICE we can also define additional firefighting agents, such as slowly-moving bulldozers, non-moving firebreaks created by bulldozers, and aircraft that are quickly-moving but less effective at extinguishing fire agents. These extensions to the forest-fire fighting environment bring up important additional issues in coordinating the effective use of heterogeneous agents and resources.

5.4 Cooperative Robotics

A motivating force in our development of MICE was to make a testbed that would be flexible enough to simulate different domains that are typically addressed in distributed problem-solving research. A second motivation was to be able to simulate coordination issues faced in actual systems that we have access to. That is, many real-world systems call for coordination between individual agents, and we would like to study the coordination issues for these systems without getting bogged down in other details. For example, our laboratory has several types of mobile robots with different capabilities. We have state-of-the-art, high-precision Cybermation mobile platforms that can carry sophisticated sensory apparatus including cameras for computer vision. These platforms have no capability to manipulate the world, however. We also have Heathkit Hero robots which move and sense imprecisely, but have manipulators through which they can pick up and move (light) objects in the world.

We are studying issues in coordination and cooperation between these two robots,

where the Hero relies on the Cybermation for sensory information, and the Cybermation relies on the Hero for manipulating the world. Depending on their individual goals, there are many ways that these systems can coordinate, and we would like to build AI software for these systems to allow them to coordinate flexibly. However, experimentally testing this software in the actual systems can be very dangerous and time-consuming: Dangerous because software bugs can lead to physical disasters such as robots colliding; and time-consuming because of the need to resolve many low-level details in controlling the robots that have nothing to do with the higher-level coordination issues.

We therefore use MICE to simulate the environment containing the Cybermation and Hero robots. We define for MICE the capabilities of both types of agents. A Cybermation has very accurate, wide-ranging sensors, and can move quickly and precisely. Heroes have limited sensory abilities and move slowly and sometimes inaccurately (they end up somewhere slightly different from where they intended), but they can link to objects in the environment and move these objects. For example, a simple task that we are trying to get our actual system to perform is to have the Cybermation find cups and wastebaskets in the environment, and to provide information to the Heroes so that the Heroes can pick up the cups and put them in the wastebaskets. In our simulation, we simulate our Heroes approaching cup objects at an appropriate orientation, linking to these objects (picking them up), carrying them to wastebasket objects, and unlinking (dropping them).

In studying this task in the real system, our work has been delayed by low-level difficulties in controlling robot motors and arms to actually perform the desired activities. The MICE simulation allows us to concentrate on the coordination issues instead, and we have implemented simple versions of the Cybermation and Hero agents in GBB. These agents currently have identical goals, and exchange messages to cooperatively pick up all of the cups. Our next stage will be to give the agents different goals. For example, Cybermation agents might have goals to map the work area, while Heroes want to pick up cups. To achieve their goals might require cooperation: The Hero might need the help of the Cybermation to locate cups and wastebaskets, while the Cybermation might need the help of a Hero to move a cup out of the way so that the Cybermation can access some otherwise inaccessible part of the area.

6 Conclusions

In essence, MICE is a shell for empirically evaluating coordination techniques in a variety of environments. MICE provides mechanisms for keeping track of agent actions and interactions in a simulated environment, and tools for evaluating the behavior of agents in the environment. It is up to the user to distill out from some task domain the essential environmental constraints and characteristics that influence coordination, and then to encode this information into MICE. Thus, MICE provides the software infrastructure for executing a simulation of some environment, but the

user must provide the domain-dependent information.[3]

As a consequence, MICE is only the first step toward our more general goal of studying multi-agent domains and extracting general principles and techniques for coordination. MICE provides the touchstone with which to evaluate alternative coordination mechanisms, and will allow us to move much more easily between simulated multi-agent environments. If we have developed agents that can coordinate well in one type of situation, we can modify MICE to evaluate the same agents in a very different situation. For example, after we had developed GBB-based agents for the predator-prey environment outlined previously, we attempted to apply these agents to the cooperative-robotics environment. Although many aspects of how agents communicate and model each other can be transferred between domains, the issues in cooperation between heterogeneous agents in cooperative robotics highlighted some assumptions about commonality that were embedded in the cooperating (and identical) predators. Insights about the transferability of coordination mechanisms across domains are much more readily developed when it is easier to move between domains and collect observations.

The initial version of MICE described in this paper has been implemented and is in use, although we already plan extensions to improve its human interface and experimental measurement capabilities. Our current research on building cooperative agents is focusing on developing a core of cooperation knowledge sources that can be part of any blackboard-based agent. Developing MICE has been a crucial first step toward this research goal It is far from clear whether there are in fact any such knowledge sources, and only by repeated implementation and evaluation can we hope to converge on them or discover that the search is hopeless. But if we fail to find such knowledge sources, will the implication be that there is no such thing as general cooperation knowledge, or will it be that a blackboard-based reasoning architecture is insufficient? Because MICE has a clean interface to the agents that act in the simulated MICE world, we can easily explore alternative architectures within MICE. Thus, in MICE we have laid the groundwork for years of experimental investigations into different facets of coordination.

References

[1] M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA, August 1985.

[2] Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray. GBB: A generic blackboard development system. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1008–1014, Philadelphia, Pennsylvania, August 1986. (Also pub-

[3] In fact, when we use GBB agents in MICE, we are using two shells. GBB provides a shell for implementing blackboard-based agents, but it is up to us to identify what the contents of a blackboard should be and what domain knowledge an agent needs. MICE provides a shell for implementing a simulation environment, but it is up to us to identify the constraints and characteristics of the simulated domain and provide these to MICE.

lished in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, pages 503–518, Addison-Wesley, 1988.).

[3] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[4] Robert F. Franklin and Laurel A. Harmon. Elements of cooperative behavior. Technical report, Environmental Research Institute of Michigan, Ann Arbor, MI 48107, August 1987.

[5] Les Gasser, Carl Braganza, and Nava Herman. MACE: A flexible testbed for distributed AI research. In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, chapter 5, pages 119–152. Pitman, 1987.

[6] Les Gasser and Nicolas Rouquette. Representing and using organizational knowledge in distributed AI systems. In *Proceedings of the 1988 Distributed AI Workshop*, May 1988.

[7] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, pages 1–64, 1987.

[8] Victor R. Lesser and Daniel D. Corkill. The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, Fall 1983. (Also published in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, pages 353–386, Addison-Wesley, 1988 and in *Readings from AI Magazine: Volumes 1–5*, Robert Engelmore, editor, pages 69–85, AAAI, Menlo Park, California, 1988).

[9] H. Van Dyke parunak. Manufacturing experience with the contract net. In Michael N. Huhns, editor, *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, chapter 10, pages 285–310. Pitman, 1987.

[10] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[11] R. Steeb, S. Cammarata, S. Narain, J. Rothenburg, and W. Giarla. Cooperative intelligence for remotely piloted vehicle fleet control. Technical Report R-3408-ARPA, Rand Corporation, October 1986.

XI. IRTPS - A Strategic Opportunity

(This report was prepared following the workshop.)

Michael Fehling
Engineering-Economic Systems
Stanford University
Stanford, CA 94305-4025
fehling@polya.stanford.edu

Bruce D'Ambrosio
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3902
dambrosio@cs.orst.edu

January 25, 1990

This memorandum presents some of our views on Intelligent, Real-Time, Problem-Solving (IRTPS)[1]. Due to limitations of time and space we attempt to restrict our discussion to remarks that we believe differ from the views expressed by others currently participating in the IRTPS initiative. If we had more time we could have been even more brief. Perhaps we could have also found a way to be more politic in expressing these views. In particular, in this memorandum we discuss the following points:

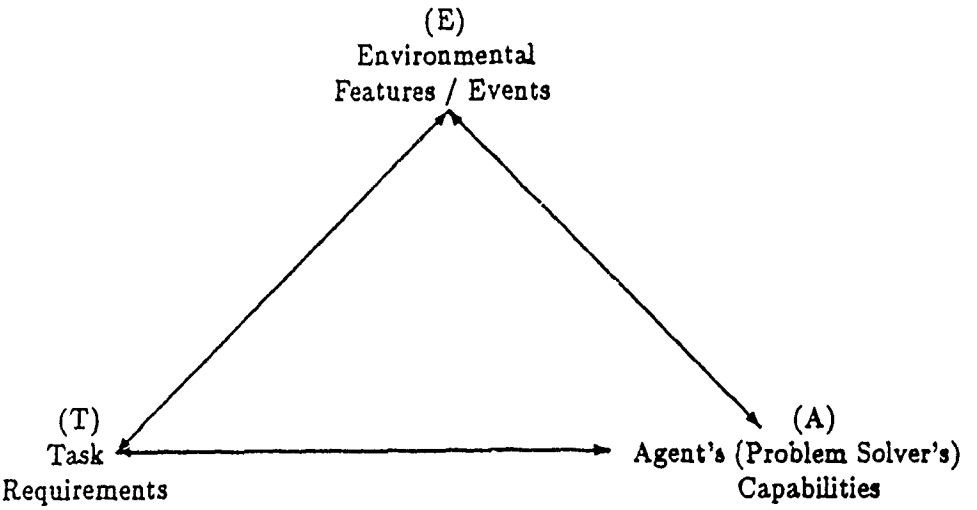
1. IRTPS is a complex of issues which involves not only a system and its environment, but also the tasks the system must perform.
2. The behavioral requirements on a IRTPS system, such as interruptibility, arise from, and must be analyzed in terms of, this triad.
3. The term *Intelligent* in IRTPS should not be taken to mean that existing problem solving methods in traditional AI are necessarily the appropriate starting point for IRTPS research.
4. IRTPS is a normative view of problem solving and autonomous agency. Decision theory, as the premier normative formal theory of rational agency, will of necessity play a central role in IRTPS.
5. A corollary of this is that uncertainty, its representation and its efficient management in dynamic situations, is a key issue for IRTPS.
6. The roles of reaction and deliberation in IRTPS are poorly understood.
7. Decision and Control theory both should be examined closely as contributors to the normative foundations of IRTPS.
8. Fundamental research and more applied research are complementary, and neither can advance without the other. A corollary of this is that existing work in the software engineering of real-time systems, as well as real-time operating systems, should be reviewed for relevance, and analysis of existing IRTPS systems should be used to test the comprehensiveness and adequacy of proposed research programs.

The Environment-Task-Agent Complex We agree that a proper perspective on IRTPS entails in part looking at the complex of issues that arise when considering the relationship between a problem-solver and its environment. (This was discussed in the Nov. workshop as the "S-E" relation.) However, we believe that the "S-E" perspective is incomplete. Specifically, the real-time aspects of IRTPS implicitly entail a commitment to some set of *tasks* to be carried out (i.e., problems to be solved) by the problem-solver. These real-time aspects of IRTPS also entail some set

[1]This memorandum was prepared as part of phases I & II of the Air Force sponsored initiative in IRTPS.

of *temporal constraints* that bound how tasks are to be completed. The temporal constraints may be part of the task requirements or they may be imposed by other conditions or processes that arise from characteristics of the environment in which the problem-solver is embedded and carries out its required tasks.

The following three way relationship summarizes this foundational view of the basic IRTPS issues:



To put this diagram into words, *IRTPS entails a complex of issues that arise when considering how an "agent's" problem-solving abilities (A) may be used to carry out some set of tasks (T) while constrained by conditions in the task environment (E), some of which are time varying.* We believe this "E-T-A" perspective more completely captures the essential IRTPS issues than does the "S-E" perspective discussed at the Nov. workshop.

IRTPS is Normative Recognizing that task requirements and environmental constraints play a fundamental role in the complex of IRTPS issues leads us immediately to the following conclusion: *IRTPS is a normative view of problem-solving.* That is, according to our E-T-A perspective, a candidate IRTPS "agent" will be measured in terms of its potential (or actual) degree of success, relative to some ideal, at carrying out the range of tasks assigned to it.

We, and subsequently others, have published analyses of "requirements" of real-time problem-solving. We have argued that such factors as timeliness, interruptability, foresight, and graceful degradation (e.g., so-called "anytime" solution methods) are important substantive issues that arise in the design or operation of real-time systems, including IRTPS systems. However, none of these factors is ascribable as a constraint that singularly applies to E, T, or A alone. Rather, these factors apparently represent bounds on the admissible relations among the features of E, T, and A. For example, a performance "requirement" such as interruptability points to a

constraint on the capabilities of a problem-solving agent that only arises if the E-T-A complex has the following characteristics:

1. Some set of events, I, is deemed sufficiently unlikely to occur that tasks associated with their occurence should not be scheduled.

2. On the other hand, should any of the events in I in fact occur, the expected cost of failing to respond is sufficiently high that some resources should be devoted to monitoring for evidence of the occurence of events in I. These monitoring tasks determine the interrupt conditions to which the agent should be responsive.

Note that these monitors are really instances of what in AI are often called "meta-level" tasks, i.e., tasks that are devoted to managing task scheduling itself. (In conventional real-time systems interrupts are often monitored by peripheral devices that operate in parallel with the processor(s) doing the primary scheduled tasks. However, this use of parallelism does not alter our account.) Note also that, as the Agent proceeds with carrying out aspects of its overall Task, and/or as new information about conditions in E is obtained, new scheduling decisions entail changes in the allocation of computational resources to the interrupt-monitoring tasks as well as all others. (In conventional real-time systems this is captured by creating levels of interrupt prioritization or interrupt "masks.")

As this example shows, one must analyze the issues of IRTPS "requirements" such as interruptability by looking at the E-T-A relationships.

It is important that future IRTPS research focus in part on teasing out the relationship between these factors, the specific characteristics of IRTPS tasks, environments, and candidate IRTPS systems, as well as the E-T-A relationships that instantiate successful achievement of such factors as interruptability.

The Intelligence in IRTPS We take a broad view of the "I" in IRTPS, Intelligence. In our view, this expresses the aim of this initiative to focus on problem-solving methods and systems that successfully carry out a broad range of real-time tasks that (a) require the use of complex information, i.e., knowledge, or perhaps (b) require adaptation, i.e., the acquisition of new knowledge, by the problem-solver during the course of task completion.

We do not agree with some workshop participants who have asserted that the focus on "intelligence" in IRTPS merely refers to an inclination to "base IRTPS problem-solving methods on traditional AI concepts and methods such as deductive planning." On the contrary, the adequacy of AI's existing computational methods for problem-solving must be scrutinized. Most extant problem-solving methods are based on assumptions that violate issues to be addressed within the E-T-A perspective. For example, nearly all AI planning methods assume that the planner has complete information at the start of planning about the effectiveness of its actions and environmental conditions, that the environment is unchanging during planning, etc. IRTPS's focus on intelligent problem-solving opens the initiative to quite poorly understood issues

that must now be addressed in a much broader context than has typically been the focus of extant AI research. We need to exercise caution in adopting the concepts and methods of conventional AI research.

Constrained Rational Agency The preceding points also lead us to conclude that proper IRTPS concepts and methods must meet standards for resource-constrained rationality that reflect (a) the need to meet task requirements, and (b) the way in which the problem-solver manages the tradeoff among critical but limited problem-solving resources. We suggest the phrase *constrained rational agency* (CRA) as a label for these issues.

One of the most challenging aspects of the IRTPS initiative will be to elicit and examine alternative theoretical paradigms as to their adequacy in explicating the issues of CRA and IRTPS. It is quite possible that theories of "rational agency" that base their operation on consistency management according to a classical (e.g., first-order) model of logical deduction provide an insufficient basis for explicating constrained rationality in general, and IRTPS in particular. Prima facie, the recent work based on formal theories of decision-making under uncertainty offer a more suitable theoretical paradigm for investigating CRA and IRTPS. Fehling, D'Ambrosio, Horvitz, Breese, and Russell (among others) have published compelling examples of the use of decision-theoretic models of CRA, including some examples that focus specifically on management of problem-solving resources in real-time tasks. These decision-theoretic concepts got relatively little attention at the Nov. workshop. We believe that this was a mistake that the IRTPS initiative cannot afford to make again.

Uncertainty in IRTPS If we are correct in our belief that the decision-theoretic is a plausible candidate paradigm for IRTPS, then this initiative must focus a significant proportion of its efforts on some topics that have received comparatively little attention by AI researchers to date.

One of these topics is real-time *reasoning about and under uncertainty.* On the one hand it seems clear that uncertainty management is a ubiquitous requirement of nearly all IRTPS tasks. On the other hand, existing computational methods for uncertainty management are typically intractable, and hence not obviously suited for use in IRTPS systems. In other words, we face a dilemma when considering the treatment of our uncertainty: uncertainty is ubiquitous, but uncertainty-management methods are intractable. To resolve this imposing dilemma, the IRTPS initiative should significantly encourage research on such issues as efficient methods for dynamically updating uncertain beliefs, uncertainty in planning and reasoning about action, and coping with uncertainty in "meta-level" control of problem-solving. Furthermore, as Ward Edwards has pointed out, the world is full of uncertainty, but it is not always necessary to represent it. We have written extensively regarding requirements for uncertainty management and will not attempt to reproduce those arguments in their entirety here. Fundamentally, the complexity of a fully quantitative treatment of uncertainty, combined with the vast extent of knowledge typically available to an agent,

preclude complete integration of the available information in all but the most trivial situations. Better understanding is needed of the problem of bounding the information (data and knowledge) and processing resources (space and time) in uncertainty management during IRTPS. We have suggested that the dimensions of this problem include:

1. Initial knowledge selection.
2. Incremental precision refinement (either through interval bounding or error estimate techniques).
3. Incremental updating in the presence of new information, including incremental reformulation of the knowledge to be brought to bear.

One easy way to start the bounding process is to replace quantitative, measure-based representations of uncertainty, with qualitative alternatives such as multi-valued logics, and other "logicist" alternatives. We believe this tactic is in error for the following reasons. First, it is yet to be demonstrated that these alternatives offer a sufficient representation of the properties of probabilistic reasoning, i.e., they are not "probabilistically coherent." Second, by ignoring measure-based information, these alternatives limit their ability to address one of the most critical issues in IRTPS, the ability to manage (i.e., represent and reason) about *tradeoffs*.

Management of a problem-solver's preferences is another important topic that arises when considering the issues of decision making under uncertainty. Unfortunately, the research literature has focused even less attention on this topic than it has on uncertainty management. Nevertheless, IRTPS clarifies the importance of preference management in intelligent agents. It seems obvious that a successful IRTPS system must be able to choose among alternative actions by trading off such things as quality or completeness of the expected result versus the anticipated time to obtain it. In order to manage these tradeoffs, the problem-solving agent must be able to dynamically manage preferences that reflect the agent's relative preferences for successfully completing its primary tasks, maintain the agent's integrity (e.g., survive), etc. Note that in a real-time task environment the agent's preferences among the most immediate conditions may change. Consider for example an autonomous robotic exploratory vehicle carrying out experiments on the surface of an alien planet. Although that agent's primary long-term objective is to complete its assigned experiments, environmental conditions that threaten the agent's continued operation (e.g., an imminent solar storm) might force it to prefer postponing an experiment in favor of seeking temporary shelter. In the AI literature, almost no attention has been given to the issues of goal creation or managing tradeoffs among alternative goals. In fact, the deeper issue of goals and their relationship to underlying preferences and desires has yet to be adequately addressed by the AI research community. Although decision theory offers a formal basis for treating these issues, the decision-theory community has also given comparatively little attention to dynamic, real-time preference management.

In light of these remarks we stress once again how unfortunate it is that the Nov. workshop failed to focus more seriously on mature paradigms that address these issues such as decision theory or stochastic control theory.

Reaction and Deliberation For the most part, methods of combinatorial search underlie AI's computational methods for *deliberation* in problem-solving. In deliberative problem-solving some form of inference such as logical deduction is employed to derive conclusions or select actions. Traditional work in planning, for example, is essentially search for a satisficing action sequence. Recently, it has become fashionable to challenge this view, as Chapman has done in his recent MS thesis, by pointing out the inherent intractability of such deliberative processes. Even the most optimistic expectations of advances in parallel hardware do not give us reason to hope that problems of real world complexity are amenable to naive search-based solutions.

Although this complexity problem for deliberation stands independent of IRTPS concerns, it is of critical importance to IRTPS. In particular, an often advanced answer to the complexity problem is "meta-level reasoning", that is, deliberative problem-solving to formulate a control strategy specifying the sequencing of reasoning and other problem-solving actions. More specifically, meta-level deliberation is employed to select problem-solving actions that are as efficient as possible while still providing a satisfactory level of task completion. Fehling, Horvitz, Breese, and Russell are the main proponents of this approach. (For the most part, they have based their meta-level mechanisms on decision-theoretic principles.)

Reliance upon meta-level deliberation may offer a partial answer. However, it alone is insufficient as a solution to the complexity problems of IRTPS. One difficulty, familiar to everyone who has written a recursive procedure, is: who controls the controller, or where do the meta levels end? A second difficulty is that, unless it is carefully crafted, meta-level reasoning can exacerbate the response-time problem. A simple and standard answer to the problem of endless metalevels of control is to use only a single meta level. While this terminates recursion, it leaves unanswered questions regarding the adequacy, flexibility, and robustness of the fixed meta-level. Certainly, this approach inadequately solves the problem of meta-level deliberation for the general case and, hence, is best viewed as more of an engineering tactic than a scientific solution to the management of problem-solving complexity.

A more recent and orthogonal alternative is the study of various partial solution techniques which can continuously refine solutions as long as time permits. These "anytime" methods may help, but again they are insufficient in general. For one thing, in general an IRTPS agent may need to engage in some deliberation to determine whether a partial answer already computed is acceptable or it is possible and worthwhile to further refine the answer. In other words, anytime algorithms still require control decisions for the use. And, if those control decisions are implemented as fixed policies at design time, then all the questions of adequacy, robustness, and flexibility arise once again.

Another obvious strategy for coping with the complexity of deliberation is "com-

pilation." In this case one introduces a reactive (non-deliberative) element that represent a fixed response to one or more anticipated problem-solving situations. Research on reactive approaches currently takes several forms. Some (e.g., Rosenschein or Subramanian) are pursuing methods for transforming a problem description which would normally be given to a deliberative mechanism, such as a theorem prover, into a combinatorial logic circuit which can compute the same output in bounded time. Others (e.g., Brooks) seek to directly engineer reactive architectures. Still others seek to place a reactive front end on a deliberative engine (Cohen, Linden) to ensure adequate response time, with a fixed policy for dividing responsibilities for taking problem-solving actions between the reactive and the deliberative components.

Overall, no one seems to have a clear understanding of the relationship between reaction and deliberation. For example, given a suitable compilation technology, is deliberation still relevant at all? When? How does compilation savings in speed impact the requirements for storage space?

We offer the following observations about the relation between reaction and deliberation in IRTPS systems:

1. Both reaction and deliberation will be essential component technologies of CRA's, and so work on purely reactive and purely deliberative strategies should continue, but with the recognition that these are only pieces, not entire solutions. Specifically, research on "anytime" and resource-bounded reasoning and decision-theoretic control of problem solving should be pushed. It is less clear that continued pursuit of simple heuristic single-level meta-level reasoning approaches will contribute substantially to IRTPS.

   ch which promises to provide mechanisms for transforming between delib-
   ve and reactive problem formulations is particularly crucial to the IRTPS
   r. Note that in addition to compilation approaches currently being
   ..d, much work in machine learning can be seen as contributing to this area. The machine learning work may actually be further along, and should be reviewed for potential relevance.

3. We believe that purely reactive solutions to most interesting problems will either be unobtainable or too large to field. Since a balance must therefore be established between the size of the reactive component and the speed of the deliberative component, rather than simply selecting one or the other, it seems likely that elements of decision theory will enter into evaluating the tradeoffs involved.

4. The most fundamental research to be performed is architectural. That is, we must understand the relationship and interaction between reaction and deliberation. No existing AI architecture adequately accounts for this interaction.

Control Theory Control theory, like decision theory, rests upon general concepts and methods that have clear prima-facie relevance to IRTPS. Building upon these

concepts, the fields of control theory and (process-)control engineering have successfully fielded an enormous amount of practical technology for real-time solution of problems in complex, dynamic environments. Also like decision theory, the principle difference from AI seems to lie in the control theoretic focus on quantitative rather than qualitative methods and in analytic, optimal solutions rather than approximate, heuristic ones. However, the IRTPS initiative would be well advised to carefully examine the potential of research and development of quantitative methods for real-time control for informing the present initiative. For example, a better understanding of work on such control-theoretic concepts as observability and state identification is likely to inform AI research on such topics as real-time diagnosis and situation assessment. In addition, certain notions like stability, convergence, and robustness play an important role in control theory for characterizing the properties of a solution to a control problem. It is quite possible that qualitative analogues of such concepts could play an important role in characterizing the adequacy of an IRTPS system for performing particular tasks. At the very least, there are most probably important lessons to be learned by exploiting the knowledge for building conventional real-time control systems to provide suggestions for building IRTPS systems.

Traditional Real-Time Computing and IRTPS Research on the software engineering and operational characteristics of non-AI, real-time software (including real-time operating systems) provides another body of work that we believe is of obvious prima-facie relevance to ITRPS. In fact, the successful examples of AI systems that exhibit true real-time performance all borrow important techniques from conventional real-time systems development. ITRPS researchers have two choices: they can find ways to apply the best ideas from conventional real-time software engineering, or they can fail to discover the relationship between IRTPS and this important body of research and thereby be forced to "reinvent many wheels."

Development Methodology/Tools versus Operational Principles Practitioners in fields such as control theory and real-time operating system design have learned that progress on system-development methods proceeds concurrently with progress on the development of new operational concepts and techniques. We expect that this will be the case for IRTPS as well. That is, the IRTPS initiative would be poorly served if too much emphasis were placed on software development at the expense of research on new IRTPS methods and architectures, and conversely.

Learning from Prior Application Experiences There is an existing base of experience in building experimental, prototype, and successfully fielded IRTPS systems. It is essential that IRTPS researchers exploit this. In particular, our own views, and the contributions we are able to make to the IRTPS initiative, benefit significantly from having had the opportunity to build a rather large and diverse array of real-time AI systems. Some of these, such as our Material Composition Manager, and our Inertial Navigation Advisor (also known as INS-FAAMS) have been fully fielded for

practical use or prototyped under full-field conditions. Careful retrospective analysis of these application-development experiences have helped evaluate, interpret, and guide our more basic research.

Frankly, we are quite dismayed at the minimal level of concern (or interest) displayed by IRTPS-initiative participants regarding the knowledge that can be derived by looking at previous attempts to develop real-world AI systems that must meet real-time performance constraints. We were further concerned that, at the Nov. workshop, there was essentially no information presented regarding prior IRTPS applications. Nevertheless, there is a significant body of work of this kind in addition to our own efforts.

IRTPS raises extremely difficult issues, ones that challenge the very foundations of AI theory and practice. We are convinced that, by carefully exploiting previous IRTPS application experience, our research community will enhance its understanding of the range of IRTPS issues, their significance, their relationship to one another, and their implications for existing conceptions of intelligent agency. Therefore, we view such retrospective analyses as a critical, high-proportion component of the IRTPS initiative at this early stage.

Summary We believe that the time is right for significant, fundamental scientific advances which can lead to fielding a new generation of highly capable IRTPS systems in future. However, we do not believe that these advances will occur as a result of simple, narrowly focused, extension of traditional AI techniques. We believe that a much broader scientific base must be established for IRTPS, along the lines outlined in this memorandum. We are disappointed and alarmed that this basic point is apparently unrecognized, or at least disregarded, by the majority of the participants at the November workshop. We believe that for the IRTPS initiative to be productive and efficient the issues we have raised must be addressed.